تاریخ دریافت مقاله: ۹۸/۰۱/۱۱
تاریخ پذیرش مقاله: ۹۹/۰۱/۲۴

بیشینه‌یابی با استفاده از مقایسه‌های نادقیق: بررسی نظری و تجربی

محمد فرشی*
استادیار، آزمایشگاه الگوریتم‌های ترکیبیاتی و هندسی، دانشکده علوم ریاضی، دانشگاه یزد، یزد، ایران
پست الکترونیکی: mfarshi@yazd.ac.ir

پریا قلی‌پور لوائی
کارشناس ارشد، آزمایشگاه الگوریتم‌های ترکیبیاتی و هندسی، دانشکده علوم ریاضی، دانشگاه یزد، یزد، ایران
پست الکترونیکی: paria.gholipour@stu.yazd.ac.ir

چکیده

بیشینه‌یابی، یکی از مسائل پایه‌ای در علم رایانه است و الگوریتم‌های متعددی وجود دارند که با استفاده از مقایسهٔ عناصر، نسبت به بیشینه‌یابی اقدام می‌کنند. در مدل‌های مرسوم، مقایسهٔ دو عنصر به‌صورت دقیق انجام می‌شود. ما مدل مقایسه‌های نادقیق را در نظر می‌گیریم: به این معنی که، اگر تفاوت مقدار دو عنصر زیاد باشد، حاصل مقایسه دقیق است؛ در غیر این صورت ممکن است نتیجه مقایسه با واقعیت منطبق نباشد. در الگوریتم‌های موجود برای بیشینه‌یابی در مدل مقایسه‌های نادقیق، ممکن است خطای ناشی از مقایسه‌های نادقیق انباشته شود و موجب خطای زیاد در خروجی نهایی گردد. می‌توان با هزینهٔ انجام تمام مقایسه‌ها، به پایین‌ترین حد ممکن خطا رسید. اما هدف، دستیابی به یک تعادل مطلوب بین تعداد مقایسه‌های انجام‌شده و میزان خطا است. با چنین شرایطی، فلدمن و همکارانش الگوریتم k-MaxFind را ارائه دادند. در این مقاله این الگوریتم پیاده‌سازی شده و تحلیل تجربی انجام‌شده، گواه بر کارایی این الگوریتم با ورودی تصادفی نسبت به داده‌های خوشه‌ای و مرتب‌شده است، نتایج نظری، برای رسیدن به خطای $k = \lceil \log\log n \rceil$ که n اندازه ورودی است، تعداد $\Omega(n^{1+1/(2^k-1)})$ مقایسه را لازم می‌داند، در حالی که بررسی این مقاله نشان می‌دهد در عمل، روی داده‌های تصادفی و روی چند توزیع دیگر، تعداد مقایسهٔ حداکثر چند برابر اندازه ورودی، برای رسیدن به این خطا کافی است. بدین جهت در عمل الگوریتم کارایی بهتری نسبت به حالت نظری دارد. تاثیر پارامترهای مختلف بر نحوه تغییر رفتار الگوریتم در «تعداد مقایسه‌های انجام‌شده» و «خطای خروجی» نیز در این مقاله بررسی می‌شوند. قبلاً بررسی عملی این الگوریتم‌ها انجام نشده است و این مقاله اولین بررسی تجربی این الگوریتم‌ها است. نتیجه این بررسی نشان از کارایی خوب این الگوریتم در کاربرد است.

واژه‌های کلیدی: الگوریتم بیشینه‌یابی، مقایسه‌های نادقیق، الگوریتم k-MaxFind، توزیع تصادفی

* نویسندهٔ مسئول

۱-مقدمه

کار بـــا داده‌ها در کامپیوتر از جنبه‌های مختلف با عدم دقت همراه است. از یک طرف، داده‌های مورد پردازش در رایانه توســـط دستگاه‌های دیجیتالی مختلف و حسگرهای متنوع جمع‌آوری می‌شـــوند که همه دارای حافظه بســـیار محدود هســـتند و لذا داده‌های جمع‌آوری شده را با تقریب ارائه می‌کنند. همچنین با توجه به دقت محدود رایانه، گاهی انجام مقایسه داده‌های بسیار نزدیک به هم که تفاوت آن‌ها در رقم‌های خارج از محدوده دقت ذخیره‌سازی داده‌ها در رایانه اســـت نیز می‌تواند با عدم دقت همراه باشد. این عدم دقت، در صورت مدیریت نشدن، می‌تواند به خطاهای برزگ در نتیجه محاسبات منجر شود و نتایجی کاملاً متفاوت با واقعیت ارائه کند. این موضوع در اســـتفاده از دستگاه‌های مرتبـــط با رایانه کـــه با دقت محدود هســـتند نیز می‌تواند اتفاق بیفتد. علاوه بر این، در بررســـی‌های نظیر بررســـی داده‌های اجتماعی که افراد در تکمیل پرســـش‌نامه‌های آن دخیل هستند، دریافت نظرات کاربران همواره با عدم دقت اســـت و همواره این عدم دقت در تحلیل داده‌های مربوطه مد نظر قرار می‌گیرد. این موضوع می‌تواند در مدل‌سازی سیســـتم‌های مختلف نیز اتفاق بیفتد و عدم توجه به وجود خطا در داده‌ها و عدم دقت محاسبات، ممکن است به مدلی با رفتار کاملاً متفاوت از سیستم اصلی منجر شود.

به‌عنوان مثال، در مرجـــع [۱] الگوریتم‌هایی تحمل‌پذیر نقص برای تراشه‌ها و شـــبکه‌ها ارائه شده است و میزان کارآیی آن نیز با شبیه‌ســـازی بررسی شده است. در این کار فرض بر این اســـت که داده‌های داده شده به الگوریتم و همچنین الگوریتم شبیه‌ســـازی کاملا دقیق است و انجام محاســـبات و مقایســـه‌ها نیز در تمام روال این الگوریتم‌ها کاملاً دقیق انجام می‌شـــود. چنین شرطی در واقعیت اتفاق نمی‌افتـــد و چه بســـا ورود یـــک خطای ناچیـــز در روند محاسبات و تجمیع این خطا در روند الگوریتم، به نتیجه‌ای کاملاً متفاوت با واقعیت منجر شود.

لذا در بحـــث طراحـــی الگوریتم، به جای اســـتفاده از مدل‌های کلاســـیک، که فرض انجام محاسبات و مقایسه‌ها به صورت دقیـــق را دارند که منطبق بر واقعیت رایانه‌های موجود نیســـت، مـــدل را به این صورت تغییـــر داده‌اند که مقایســـه داده‌های نزدیک به هم، بـــا توجه به دقت محدود ذخیره‌ســـازی می‌تواند با خطا انجام شـــود و در طراحی الگوریتـــم در این مدل، به امکان وجود خطا در مقایســـه‌ها توجه شده اســـت تا از هم‌افزایی خطاهای کوچک و منجر شدن آن به خطای زیاد در جواب الگوریتم جلوگیری شود. طراحـــی الگوریتم‌هـــا در این مدل، عملاً ســـخت‌تر از مدل مرســـوم طراحی الگوریتم است، اما به دلیل انطباق بیشتر آن با رایانه، الگوریتم‌های طراحی شده در این مدل مطمئناً کاربرد بیشتری در عمل خواهند داشت.

یائو و یائو [۱] می‌خواســـتند بیشـــینه عنصر را در یک شـــبکه مرتب از مقایســـه‌گرها پیدا کنند. هر مقایسه‌گر یک دستگاه ۲-ورودی۲-خروجی است که قادر به مرتب‌سازی دو عدد می‌باشد. اگر e مقایسه‌گر معیوب باشند؛ یعنی بدون انجام مقایسه، ورودی‌ها را مستقیماً به‌عنوان خروجی در نظر بگیرند، آنگاه برای بیشـــینه‌یابی، $(e+1)(n-1)$ مقایسه لازم و کافی است.

راویکومار و همکارانش [۲]، با استفاده از پرس‌وجوهای مقایســـه‌ای که پاســـخ «بله» یا «نـــه» را دریافت می‌کردند، بزرگ‌ترین عـــدد در مجموعه‌ای از n عـــدد صحیح متمایز را یافتنـــد. آن‌ها نشـــان دادند، برای این بیشـــینه‌یابی، اگر حداکثر e پرس‌وجو پاسخ اشتباه دریافت کند، $(e+1)n-1$ مقایســـه لازم و کافی است. با فرض محدودیت بیشتر، که خطا فقط به پاسخ‌های «نه» محدود شود و پاسخ‌های «بله» مطمئناً درســـت باشد، آن‌ها ثابت کردند $2n+2e-4$ مقایسه کافی است. آن‌ها با گسترش مسئله برای خطاهای دلخواه، نشـــان دادند که $O(en)$ مقایســـه لازم و کافی است. درنهایت فلدمن و همکارانش [۳] مسئله بیشینه‌یابی در مدل مقایسه‌های نادقیق را، با ارائه الگوریتم قطعی k-MaxFind حل کردند.

در ایـــن مقاله، ابتـــدا الگوریتم فلدمن ارائه می‌شـــود و

سپس عملکرد این الگوریتم روی داده‌های تصادفی و با برخی توزیع‌های دیگر بررسی می‌شود. مطالعه تجربی عملکرد الگوریتم‌ها، یک رویکرد پژوهشی است که طی سال‌های اخیر به آن بهای بسیار داده می‌شود. دلیل این امر این است که اولاً بسیاری از الگوریتم‌های ارائه شده به صورت نظری، به قدری پیچیده است که امکان پیاده‌سازی و استفاده از آن‌ها حتی سال‌ها پس از ارائه الگوریتم وجود نداشته است و عملاً این الگوریتم‌ها در محدوده نظری باقی مانده‌اند و وارد بحث کاربرد نشده‌اند. ثانیاً در برخی الگوریتم‌ها یک شکاف عمیق بین نتایج تحلیل زمان اجرای الگوریتم و آنچه در عمل اتفاق می‌افتد وجود دارد. در برخی الگوریتم‌ها، تحلیلی نظری حاکی از سرعت اجرای بالای الگوریتم است در حالی که پس از پیاده‌سازی و مقایسه عملکرد الگوریتم با الگوریتم‌های دیگر حل مسئله، در عمل چنین کارکردی مشاهده نمی‌شود. دلایل مختلفی نیز برای این امر وجود دارد که به‌عنوان نمونه می‌توان به فاصله بین مدل تحلیل نظری الگوریتم‌ها و رایانه‌های واقعی به دلیل ساده‌سازی مدل و همچنین حذف ثابت‌ها در تحلیل نظری الگوریتم‌ها اشاره کرد.

در این مقاله، تلاش شده است که عملکرد عملی الگوریتم محاسبه بیشینه روی داده‌هایی تصادفی بررسی و میزان کارایی آن‌ها در عمل مشخص شود. بررسی تجربی ما نشان می‌دهد، تعداد مقایسه‌های لازم برای رسیدن به یک دقت مشخص، در عمل بسیار کمتر از تعداد بیان شده در نتایج عملی است. این الگوریتم روی داده‌هایی با اندازه بین ۱ میلیون و ۱۰ میلیون و با توزیع‌های یکنواخت، مرتب‌شده و خوشه‌ای بررسی شده است و تعداد مقایسه‌های انجام شده در الگوریتم و خطای حاصل شده در عمل برای این داده‌ها به دست آمده است. نتایج حاصل شده، حاکی از تفاوت بسیار زیاد بین نتایج عملی و نتایج نظری در خصوص این الگوریتم است. به‌عنوان مثال، برای یک میلیون داده با توزیع یکنواخت، برای رسیدن به خطای ۲، کران‌های نظری تعداد بیش از ۲ میلیارد مقایسه را لازم می‌داند، در حالی که طبق بررسی تجربی انجام شده، الگوریتم‌ها با انجام حداکثر ۳ میلیون مقایسه، بیشینه را با خطای حداکثر ۱٫۵ به دست می‌دهد. این رفتار در داده‌های با توزیع‌های دیگر نیز مشاهده می‌شود. با افزایش اندازه ورودی، این تفاوت نیز بیشتر می‌شود. این بررسی نشان می‌دهد که این الگوریتم می‌تواند در عمل بسیار بهتر از نتایج نظری عمل کند و می‌تواند در کاربردهای بسیاری که عملاً داده‌های ورودی طبق یک توزیع تصادفی توزیع شده‌اند، مورد استفاده موفق قرار گیرد.

طبق اطلاعات ما، تاکنون بررسی عملکرد عملی این الگوریتم‌ها صورت نگرفته است و این اولین کار در بررسی عملی این الگوریتم‌ها است.

در ادامه بعد از بیان مفاهیم موردنیاز، این الگوریتم، نتایج نظری حاصل از تحلیل آن و پیاده‌سازی عملکرد الگوریتم مطرح می‌شوند.

۲- مفاهیم و اصطلاحات

فرض کنید مجموعه $X = \{x_1, x_2, \ldots, x_n\}$ به‌عنوان ورودی داده شده است، به‌طوری‌که هر x_i در آن، دارای مقدار نامعلوم $val(x_i)$ باشد. x_i نشان‌دهنده عنصر iام است که گاهی اوقات، به‌عنوان مقدار آن عنصر نیز استفاده می‌شود. به‌عنوان مثال $x_i < x_j$ می‌تواند بیان‌گر $val(x_i) < val(x_j)$ نیز باشد. دقت کنید که $|X| = n$ است.

در این مقاله، ما یک مدل ساده از مقایسه‌های نادقیق را در نظر می‌گیریم. فرض کنید می‌خواهیم دو عنصر x_i و x_j را باهم مقایسه کنیم. در مدل مقایسه‌های نادقیق، $\delta > 0$ وجود دارد که اگر $|val(x_i) - val(x_j)| > \delta$، آنگاه نتیجهٔ مقایسه دقیق و درست است، در غیر این صورت نتیجهٔ مقایسه، نادقیق است و نمی‌توان تشخیص داد که کدام عنصر بزرگ‌تر می‌باشد. با چنین تعریفی ممکن است پیدا کردن یک بیشینه دقیق و درست، غیرممکن باشد. فرض کنید مقادیر همهٔ عناصر ورودی در حدود δ باشد، پس تمامی مقایسه‌ها به‌صورت نادقیق انجام می‌شوند. درنتیجه

در بدتریـــن حالت هرکـــدام از عناصـــر ورودی می‌تواند عنصر بیشینه باشد. با این حال ممکن است شناسایی یک بیشینه تقریبی، شـــدنی باشد. عنصر $x_i^* \in X$ یک بیشینه تقریبی، یا k–بیشـــینه برای همه X_iها اســـت، اگر بتوان آن را به‌گونه‌ای یافت که عبارت $|val(x_j) - val(x_i^*)| \le k\delta$ بـــرای هر $x_j \in X$ و برخی‌kها (ترجیحاً کوچک و طبیعی) برقرار باشـــد. دقت کنید هدف اصلی برای یافتن چنین x_i^* ای، به حداقل رســـاندن تعداد مقایسه‌های انجام‌شده است. منظور از خطا، کمینهٔ مقدار k به صورتی است که خروجی الگوریتم در نامساوی فوق صدق کند.

بـــدون کاســـتن از کلیت مســـئله، مـــا $\delta = 1$ را در نظر می‌گیریم. چراکه مســـئله با $\delta > 0$ دلخواه، معادل با $\delta = 1$ و ارزش ورودی x_i / δ اســـت. برای مقایســـه دو عنصر x و y، اگر مقایســـه‌گر ادعا کند x از y بزرگ‌تر است؛ یعنی $x \ge y - 1$ باشد، آنگاه می‌گوییم x، y را شکست داده است. به‌ســـادگی نیز می‌توانیم بگوییم y از x شکســـت خورده است. به‌طور مشابه x از y، k–بزرگ‌تر است، بدین معنی که $x \ge y - k$ باشد. یک عنصر k–بیشینه از یک مجموعه است، اگر آن عنصر k–بزرگ‌تر از همه عناصر در مجموعه باشد. اگر مجموعه به صراحت مشخص نشده باشد، آنگاه ما به مجموعه‌ای از تمام عناصر ورودی مراجعه می‌کنیم. یک مجموعه، مجموعهٔ k–بیشینه است، اگر یک زیرمجموعه از همه عناصر باشـــد و شامل حداقل یک K–بیشینه (یعنی عنصر از ارزش حداقل $x^* - k$) باشد.

لازم بـــه ذکر اســـت، اگر دو عنصر قبلاً باهم مقایســـه شده باشـــند و در روند اجرای الگوریتم احتیاج به مقایسه مجدد همین دو عنصر باشد، مقایسه دوباره انجام می‌شود و از نتایـــج مقایســـه انجام‌شـــده در مراحل قبل اســـتفاده نمی‌شـــود. دقت کنید تمامی لگاریتم‌های این مقاله در پایه دو هســـتند. برای راحتـــی کار، گاهی اوقـــات خطای گرد کـــردن، در نظر گرفته نمی‌شـــود. اگر کف و ســـقف اعداد تاثیر جزئی روی کران‌ها داشته باشند، در محاسبات لحاظ نمی‌شوند.

۳- الگوریتم قطعی برای بیشینه‌یابی

فرض کنید X* بیشـــینه دقیق اعداد ورودی باشد. دقت کنید اگر چندین عنصر با بیشینه مقدار وجود داشته باشند یکی از آن‌ها به دلخواه انتخاب می‌شود. هدف یافتن عنصر بیشینه، تنها با استفاده از مقایسه‌های جفت‌جفت است.

بـــا توجه به این‌که هیچ الگوریتم قطعی بیشـــینه‌یابی با خطای کمتر از ۲ وجود ندارد (قضیه ۳-۲ [۱])، الگوریتم 2-MaxFind با انجام حداکثر $3n^{3/2}$ مقایسه یک عنصر بـــا مقدار حداقل 2-x* را برمی‌گرداند (لم ۳-۳ [۱]). در نظر داشـــته باشـــید که s>1 در الگوریتم یـــک پارامتر با مقدار دلخواه اســـت که اگر مقدار آن مشخص نشود، به‌صورت پیش‌فرض $\lceil \sqrt{n} \rceil$ در نظر گرفته می‌شود. منظور از تورنمنت بـــا گردش نوبت، انجام تمام $n(n-1)/2$ مقایســـه، ســـپس مرتب‌سازی عناصر با توجه به تعداد بردهایشان به منظور یافتن برنده نهایی تورنمنت است.

الگوریتم 2-MaxFind یک عنصر بیشـــینه را با خطای ۲ می‌یابـــد. ایـــن الگوریتم را طوری تغییـــر می‌دهیم که به جای یک عنصر بیشـــینه، مجموعه‌ای شامل ۱- بیشینه را برگرداند. این الگوریتم که همان الگوریتم 1-Cover اســـت، بجای $x^* - 2$، یک مجموعهٔ۱- بیشینه را برگرداند.

کلید گام بازگشـــتی از الگوریتم بیشـــینه‌یابی با خطای کلی، الگوریتم 1-Cover اســـت. الگوریتم 1-Cover با تنظیم پارامتـــر $s = \lceil 2\sqrt{n} \rceil$ در الگوریتـــم 2-MaxFind به دســـت می‌آید. با ایـــن تفاوت که خروجی آن اجتماعی از xها، در گام ۳ در هـــر بار تکرار حلقه، به‌عـــلاوه مجموعه از اندازه $\lceil \log(s) \rceil$ که ۱-بزرگ‌تر از عناصر نامزدهای باقی‌مانده در گام ۵ است.

الگوریتم 2-MaxFind

۱. ورودی: $X = \{x_1, x_2, \mathrm{K}, x_n\}$

۲. خروجی: یک عنصر با مقدار حداقل $x^* - ۲$

۳. همهٔ x_iها را به‌عنوان نامزد برچسب‌گذاری کن.

٤. تـــا زمانی که بیش از عنصر نامزد وجود دارد مراحل ۳ و ٤ را انجام بده:

۵. زیرمجموعـــهٔ دلخواه S تایـــی از عناصر نامزد را انتخاب کن و روی آن‌ها یک تورنمنت با گردش نوبت برگزار کن.

۶. x را با همهٔ عناصر نامزد مقایســـه کن و عناصری که از x باخته‌اند را حذف کن.

۷. روی عناصر نامزد باقی‌مانده (که حداکثر s تا هستند)، یک تورنمنت با گردش نوبـــت برگزار کن و عنصر با بیشترین تعداد برد را برگردان.

الگوریتم K-MaxFind

ورودی: مجموعه $X = \{x_1, x_2, \mathrm{K}, x_n\}$

خروجی: یک k–بیشینه

۱.اگر $n \le 81$ بود، خروجـــی الگوریتـــم 2-MaxFind را روی عناصر ورودی برگردان.

۲.ابتدا n عنصـــر ورودی را در مجموعه‌های $S_1, \ldots, S_t$ به‌صورت مســـاوی افراز کن، به‌طوری‌که هرکدام از آن‌ها دارای اندازه $r = \max\{81, 4 \cdot n^{8/(3\cdot 2^k-4)}\}$ باشد.

۳.الگوریتم 1-Cover روی هر S_i به منظور بازیافتن یک مجموعه T_i، که 1–بزرگ‌تر از S_i باشد، فراخوانی کن.

۴. $(k-1)$–MaxFind $(U_{i=1}^{T_i} T_i')$ را بازگـــردان. (این فرآیند بازگشتی در 2-MaxFind متوقف می‌شود).

۳-۱- نتایج نظری

کران‌هـــای نظری تعداد مقایســـه‌ها در الگوریتم قطعی بیشینه‌یابی، به‌طور خلاصه در جدول ۱ آورده شده است. نحوه محاسبه این کران‌ها نیز، در بخش ۳ مقاله [۱] به‌طور کامل شرح داده شده است.

نتیجه نظـــری ۲-۱-۱: یـــک الگوریتم بیشـــینه‌یابی با استفاده از $O(n)$ مقایسه و خطای $\log\log n$ وجود دارد.

قضیه ۲-۱-۲: هر الگوریتم قطعی بیشینه‌یابی مانند A با خطای k نیاز به انجام $\Omega(n^{1+1/(2^k-1)})$ مقایسه دارد.

پس با بیان قضیه ۱ می‌توان دریافت که نتیجه ۱ اساساً محکم است.

نتیجه نظری ۲-۱-۳: اگر A یک الگوریتم بیشـــینه‌یابی قطعی باشـــد که $O(n)$ مقایســـه را انجام می‌دهد، آنگاه A

جدول ۱: بررسی اجمالی کران‌های تعداد مقایسه‌ها در الگوریتم‌های قطعی با تنظیم $a = 1$ و خطای k

	کران‌های بالا			کران پایین
بیشینه‌یابی	$k = 2$	k	$k = \log\log n$	
	$2n^{3/2}$	$O(n^{1+1/(3\cdot 2^{k-2}-1)})$	$O(n)$	$\Omega(n^{1+1/(2^k-1)})$

حداقل دارای خطای $\log\log n - O(1)$ است.

۴- پیاده‌سازی الگوریتم بیشینه‌یابی قطعی

با توجه به این که الگوریتم k-MaxFind تنها با استفاده از مقایســـه کردن عناصر ورودی و بـــدون نیاز به دانش درباره چگونگی رفتار توابع مورد استفاده در این الگوریتم، یک k–بیشینه متناســـب با ورودی را می‌یابد، استفاده از این الگوریتم جهت رســـیدن به تعادل قابل قبول بین میزان خطا و تعداد مقایســـه‌های انجام‌شده، کافی است. از همین رو و بـــا توجه به این‌که الگوریتـــم 1-Cover یک زیرروال برای الگوریتم k-MaxFind است و 2-MaxFind، نیز بعد از کاهش اندازه ورودی فقط یک‌بار جهت پیدا کردن بیشـــینه نهایی مورد استفاده قرار می‌گیرد، پس‌نیازی به پیاده‌سازی جداگانـــه و تحلیل رفتار آن‌ها نیســـت. در این مقاله به‌طور خاص درباره پیاده‌ســـازی الگوریتم k-MaxFind فارغ از چگونگی رفتار توابع مورد استفاده در آن و تغییر رفتار این تابع نسبت به تغییر پارامترهای مختلف پرداخته می‌شود. بنابراین از این پس هر جا از کلمه «الگوریتم» اســـتفاده شد مقصودمان الگوریتم k-MaxFind است.

در پیاده‌ســـازی انجام‌شـــده، عناصـــر ورودی از بازه $X = [-100000.000, +100000.000]$ به تصادف انتخاب می‌شـــوند. حالتـــی را در نظر بگیرید که دو عنصر یکســـان باشـــند. مقایسه نادقیق این دو عنصر خطایی را ایجاد نمی‌کند. پس برای دستیابی به خطای پیاده‌سازی نزدیک واقعیت، نیاز به استفاده از اعداد تصادفی است که تکراری نباشند.

برای تحلیل بهتر الگوریتم، تمامی خروجی‌ها، میانگین ۵۰ بار اجرای الگوریتم روی ورودی‌های متفاوت اســـت. فقط به منظور تحلیل آســـان‌تر داده‌های خروجی در بخش

۳-٤ از ورودی حالت اول استفاده شده است. معمولا اجرای الگوریتم روی یک مجموعه ورودی همواره یک جواب دارد. اما در مورد این الگوریتم به دلیل مقایسه نادقیق که مقدار آن تصادفی است، این اتفاق رخ نمی‌دهد. پس برای ۵۰ بار اجرای الگوریتم دو روش امکان‌پذیر است:

روش اول: آرایه‌ای از اندازه مورد نیاز با عناصر تصادفی ایجاد می‌شود. این آرایه را به منظور اجراهای مجدد الگوریتم نگه‌داری کنید. پس الگوریتم ۵۰ بار روی همین آرایه اجرا می‌شود.

روش دوم: مشابه روش اول آرایه‌ای از اندازه موردنیاز با عناصر تصادفی ایجاد می‌شود. الگوریتم با چنین ورودی اجرا می‌شود. برای اجرای مجدد الگوریتم آرایه جدید با عناصر تصادفی تولید کنید. پس برای ۵۰ بار اجرا الگوریتم از ۵۰ آرایه تصادفی متفاوت استفاده می‌شود.

در اینجا برای اجرای الگوریتم، هر دو حالت ورودی مورد بررسی قرار گرفته است. همان‌طور که انتظار می‌رفت، حالت دوم به دلیل تنوع بیشتر الگوهای ورودی، موجب خطای جزئی بیشتری نسبت به حالت اول در برخی از اجراها شد. اما با توجه به انتخاب دلخواه عناصر در تورنمنت با گردش نوبت و ماهیت تصادفی بودن نتیجه در مقایسه نادقیق، به‌طور کلی تفاوت قابل ملاحظه‌ای بین دو حالت ورودی دیده نشد.

پارامترهایی که تغییر رفتار آن‌ها در این پیاده‌سازی بررسی شده‌اند: اندازه آرایه ورودی (n)، آستانه تفاوت (δ) و خطای نظری (k) هستند. s نیز تعداد عناصری است که در مراحل اجرای الگوریتم، روی آن‌ها تورنمنت با گردش نوبت برگزار می‌شود. پس s پارامتری است که تغییرات آن نیز در روند اجرای الگوریتم موثر است. نکته جالب توجه در فرایند پیاده‌سازی، تاثیری است که این پارامترها روی تعداد مقایسه‌های انجام‌شده و خطای خروجی الگوریتم دارند.

٤-۱- جزئیات اجرا

الگوریتم در نرم‌افزار DEV-C++ نسخه 5.6.3 پیاده‌سازی و اجرا شده است. سپس نتایج حاصل، با استفاده از نرم‌افزار متلب نسخه R2013a در قالب نمودار ترسیم شده‌اند. جزئیات سخت‌افزاری رایانه‌ای که به وسیله آن پیاده‌سازی‌ها انجام‌شده است، از قرار زیر می‌باشد:

2.50GHz Processor: Intel(R) Core(TM) i5-4200M CPU @ 2.500 GHz

Installed memory (RAM): 6 GB

System type: 64-bit Operating System, x64-based processor

Windows edition: Windows 8 Pro

٤-۲- نتایج تجربی

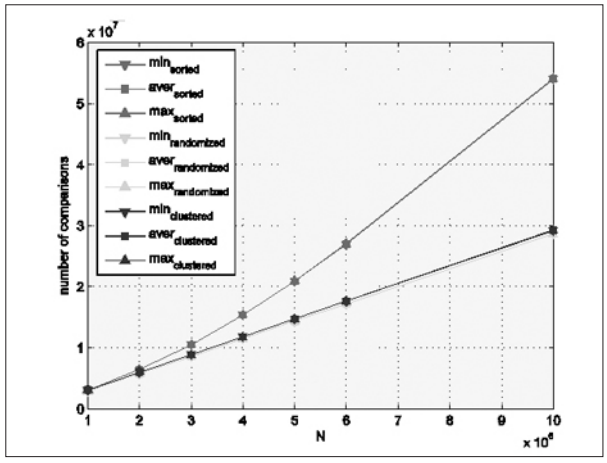
در تحلیل رفتار الگوریتم، دو پارامتر سنجیده خواهند شد.

اولین پارامتر، تعداد مقایسه‌های انجام‌شده در الگوریتم برای پیدا کردن یک k-بیشینه است. در شمارش تعداد مقایسه‌های انجام‌شده توسط الگوریتم، مجموع تعداد مقایسه‌های دقیق و نادقیق فارغ از معنا و مفهوم آن‌ها، در نظر گرفته خواهد شد. منظور از پیچیدگی الگوریتم نیز همین تعداد مقایسه‌های انجام‌شده توسط الگوریتم است (نه زمان اجرای الگوریتم).

دومین پارامتر خطای حاصل از پیاده‌سازی در خروجی نهایی الگوریتم، با نماد k' می‌باشد. اگر x_{real} مقدار بیشینه واقعی و x^* خروجی الگوریتم باشد آنگاه مقدار k' از طریق فرمول $(x_{real}-x^*)/\delta$ محاسبه می‌شود.

در این بخش درباره جزئیات نتایج پیاده‌سازی با درنظرگرفتن این‌که الگوریتم همواره تضمین می‌کند خطای پیدا کردن یک k-بیشینه (k')، با انواع توزیع‌های داده‌ای، در عمل به مراتب از خطای به‌دست آمده از نتایج نظری (یعنی k) کمتر است، می‌پردازیم.

در پیاده‌سازی، ورودی الگوریتم داده‌های علامت‌دار با ۳ رقم اعشاری است که از بازه $X=[-100000.000,+100000.000]$ انتخاب می‌شود تا بتوان با ایجاد الگوهای متنوع و محدود به بازه مشخص، جزئیات تغییر رفتار الگوریتم نسبت به



شکل ۱: نتایج حاصل از اجرای الگوریتم با ورودی با توزیع تصادفی، مرتب‌شده و خوشه‌ای با تنظیم $\delta=1$ و $k=\lceil \log\log n \rceil$ و تاثیر آن بر تعداد مقایسه‌های انجام‌شده

توزیع‌های مختلف از داده‌ها را، بهتر مشاهده و تحلیل کرد. بـــرای این کار عملکرد الگوریتم نســـبت به ۳ نوع توزیع از داده‌های ورودی مورد بررسی قرار گرفته شده است.

توزیـــع تصادفـــی: به تعداد انـــدازه آرایه‌های ورودی الگوریتم، اعداد به‌صورت تصادفی طوری تولید می‌شوند که علاوه بر این‌که در بازه مفروض قرار می‌گیرند، تکراری نیز نباشند.

توزیع مرتب شـــده: بعد از تولید آرایه‌های تصادفی و بررسی رفتار الگوریتم نســـبت به توزیع تصادفی داده‌ها، همان آرایه‌ها مرتب خواهند شد و در اختیار الگوریتم برای بررســـی مجدد تغییر رفتار آن، نسبت به این توزیع جدید مرتب شده قرار می‌گیرند.

توزیـــع خوشـــه‌ای: برای تولید چنیـــن داده‌هایی، ابتدا $\sqrt{n}$ خوشه با هسته تصادفی ایجاد می‌شود. از هر خوشه نیز به تصادف $\sqrt{n}$ عدد غیرتکراری انتخاب می‌شـــود. در انتها اجتماع این اعداد، آرایه‌ای از اندازه n با داده‌های $\sqrt{n}$ -خوشه‌ای ایجاد می‌کند.

با توجه به شکل (۱)، داده‌ها با توزیع تصادفی نسبت به سایر توزیع‌ها تعداد مقایسه‌ها کمتری را تولید کرده است. پس از این به بعد برای تحلیل رفتار الگوریتم، از ورودی با توزیع تصادفی استفاده خواهیم کرد.

همان‌طور که در جدول (۲) مشاهده می‌کنید خطا نسبت

جدول ۲: نتایج حاصل از اجرای الگوریتم با ورودی با توزیع تصادفی، مرتب‌شده و خوشه‌ای با تنظیم $\delta=1$ و $k=\lceil \log\log n \rceil$

	n	خطا یا k'			تعداد مقایسه‌های انجام‌شده		
		min	aver	max	min	Aver	max
توزیع تصادفی	۱×۱۰^۶	۰	۰.۴۵۹۸	۱.۴۹	۲۸۵۹۶۵۶	۲۸۶۴۴۸۸	۲۸۶۹۱۹۲
	۶×۱۰^۶	۰	۰.۵۳۱۲	۱.۲۹	۱۷۱۷۶۲۴۴	۱۷۱۸۸۰۸۰	۱۷۱۹۹۹۷۳
	۱×۱۰^۷	۰	۰.۴۲۷۸	۱.۱۹۳	۲۸۶۳۳۳۱۷	۲۸۶۵۱۱۶۶	۲۸۶۶۳۹۹۰
توزیع مرتب‌شده	۱×۱۰^۶	۰	۰.۴۱۸۵	۱.۳۲۱	۲۹۵۹۴۹۴	۲۹۶۸۰۴۶	۲۹۷۵۵۳۰
	۶×۱۰^۶	۰	۰.۴۶۶۶	۱.۳۴۲	۲۶۹۲۲۵۰۷	۲۶۹۵۶۵۹۰	۲۶۹۹۳۳۰۰
	۱×۱۰^۷	۰	۰.۵۲۷۴	۱.۵۴	۵۴۰۱۶۴۹۹	۵۴۰۵۰۹۷۶	۵۴۰۷۲۸۴۹
توزیع خوشه‌ای	۱×۱۰^۶	۰	۰.۴۰۳۱	۱.۳۹۷	۳۰۶۸۷۱۷	۳۰۷۵۸۱۱	۳۰۸۲۴۵۲
	۶×۱۰^۶	۰.۰۱۲	۰.۴۱۰۳	۱.۱۶۳	۱۷۵۹۰۷۳۳	۱۷۶۰۶۱۷۴	۱۷۶۱۹۰۴۱
	۱×۱۰^۷	۰.۰۲۵	۰.۴۶۴۵	۱.۰۵۴	۲۹۱۹۱۳۶۷	۲۹۲۱۹۳۴۶	۲۹۲۵۲۹۴۷

بـــه انواع توزیع داده‌های ورودی، تغییر محسوســـی نکرده است.

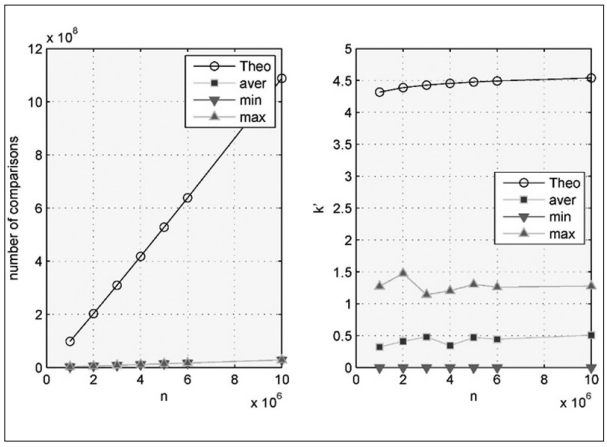
نتیجـــه جالب و حائـــز اهمیت در فرایند پیاده‌ســـازی، گواه بر کارایی الگوریتم با ورودی تصادفی در مقایسه با داده‌های خوشه‌ای و مرتب‌شده است.

٤-۳- مقایسه نتایج نظری و پیاده‌سازی

با توجه به کارایی داده‌های ورودی با توزیع تصادفی، در ادامه نشان می‌دهیم که الگوریتم با ورودی تصادفی در عمل به‌مراتب رفتار بهتری نســـبت به حالت نظری دارد. با تنظیم $\delta=1$ و $k=\log\log n$ نتایج زیر حاصل شد.

همان‌طور که در شکل (۲) مشـــاهده می‌کنید تصادفی بودن به‌طور قابل ملاحظه‌ای باعث کاهش تعداد مقایسه‌های انجام‌شـــده و بهبود خطا شـــده اســـت. دقت کنید که حتی بیشـــینه مقدار خطای ورودی تصادفی نیز اختلاف فاحش با مقـــدار خطای نظری دارد. الگوریتم با ورودی تصادفی در عمـــل به‌مراتب رفتار بهتری نســـبت بـــه حالت نظری دارد.

٤-٤-تاثیر پارامترهای مختلف بر الگوریتم



شکل ۲: مقایسه نتایج نظری با نتایج حاصل از اجرای الگوریتم با توزیع تصادفی، با تنظیم δ =۱ و k = ⌈loglogn⌉

در این بخش به بررسی تاثیر پارامترهای k، δ، n و s بر نحوه تغییر رفتار الگوریتم در «تعداد مقایسه‌های انجام‌شده» و «خطای خروجی» می‌پردازیم. اندازه مجموعه ورودی می‌باشد که مقادیر ۱، ۲، ۳، ٤، ٥، ٦ و ۱۰ میلیون است.

لازم به ذکر است تغییر اندازه مجموعه ورودی X از ۲۰ میلیون داده به ۲۰۰ میلیون داده، فقط موجب تنوع بیشتر الگوهای ورودی و به تبع آن افزایش احتمال مشاهده الگوهایی با خطای بیشتر از خطای میانگین شد.

تاثیر پارامتر k

k خطای حاصل از نتایج نظری با مقدار $\log\log n$ است.

این پارامتر در افراز کردن آرایه ورودی نیز شرکت دارد. نحوه تغییر رفتار الگوریتم برای مقادیر K=3 (کوچک‌ترین مقدار برای شروع الگوریتم)، ٤، $k=\lceil \log\log n \rceil$ (که به ازای تمام مقادیر ورودی ٥ است)، ٦ و ۱۰ بررسی شد.

نتیجه شهودی قابل استنباط از شکل (۳)، گواه بر خطی بودن تعداد مقایسه‌های انجام‌شده توسط الگوریتم به ازای مقادیر مختلف k است. این نتیجهٔ قابل انتظار، تاییدی بر نتیجه ۲-۱-۱ می‌باشد.

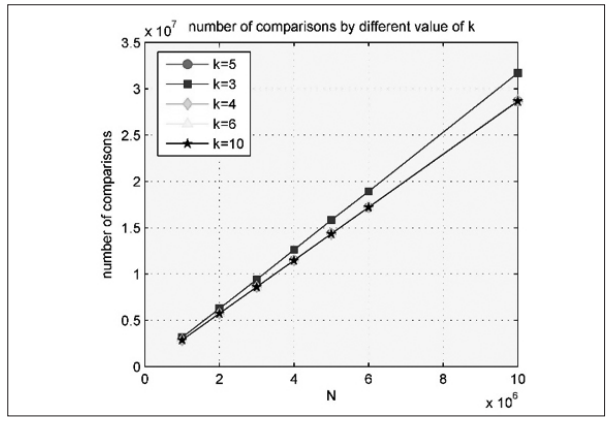
نتیجه دوم با نگاه کلی به رفتار تمام مقادیر k حاصل می‌شود. همان‌طور که در شکل (۳) مشاهده می‌کنید نمودار مربوط به $k=3$ بیشترین تعداد مقایسه بعد از آن $k=4$ و برای سایر مقادیر k نیز با افزایش میزان k شاهد افت میزان افزایش تعداد مقایسه‌ها هستیم. پس می‌توان نتیجه گرفت؛ هرچقدر میزان k افزایش پیدا می‌کند، تعداد مقایسه‌های انجام‌شده توسط الگوریتم کاهش پیدا می‌کند. اما آیا این میزان کاهش برای همه مقادیر k یکسان است؟ نکته جالب توجه زمانی است که الگوریتم به $k=\lceil \log\log n \rceil$ می‌رسد، دیگر از کاهش چشم‌گیر میزان تعداد مقایسه‌ها خبری نیست.

با نگاهی دقیق‌تر به شکل (۳) می‌توان دریافت که، با وجود خطی بودن همه نمودارها، در شیب، تفاوت وجود دارد. شیب نمودار مربوط به $k=3$ به مراتب از سایر مقادیر k بیشتر است. روند کاهش شیب تا رسیدن به $k=\lceil \log\log n \rceil$ ادامه دارد. اما بعد از آن روند کاهش شیب نمودارها کند می‌شود. از نگاهی دیگر می‌توان گفت افزایش میزان k بعد از $k=\lceil \log\log n \rceil$ باعث کاهش قابل ملاحظه‌ای در تعداد مقایسه‌ها نمی‌شود. پس در آخر نیز می‌توان نتیجه گرفت؛ $k=\lceil \log\log n \rceil$ یک کران برای کاهش تعداد مقایسه‌ها است. این نتیجه تاییدی بر نتیجه ۲-۱-۳ است.

لازم به ذکر است، کلیه نتایج این قسمت برای تمامی مقادیر δ بررسی شده است و همگی آن‌ها برقرار هستند. فقط برای ترسیم نمودار به‌صورت دلخواه از $\delta=1$ استفاده شده است.

تاثیر پارامتر δ

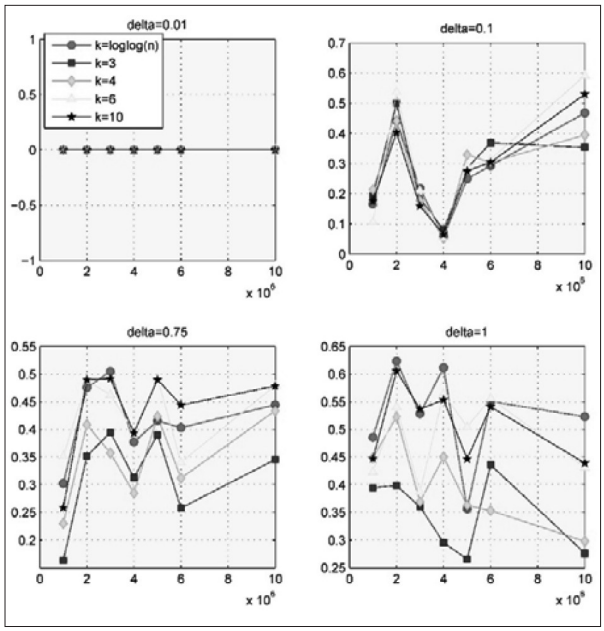
مقدار δ، آستانه تفاوت است. زمانی که میزان اختلاف دو عدد در حدود δ باشد مقایسه به‌صورت نادقیق انجام می‌شود، در غیر این صورت مقایسه به‌صورت دقیق انجام خواهد شد. مقدار δ نیز از ۰,۰۱، ۰,۱، ۰,۲۵، ۰,۵، ۰,۷۵ و ۱ تغییر می‌کند. نتیجه انجام یک مقایسه نادقیق روی دو عنصر نزدیک به هم، انتخاب یکی از آن دو عنصر به تصادف است، چراکه قدرت تشخیص عنصر بزرگ‌تر را نداریم. در پیاده‌سازی این فرایند با تولید یک عدد طبیعی به تصادف و محاسبه باقی‌مانده آن به ۲ آغاز می‌شود.



شکل ۳: تاثیر پارامتر k بر تعداد مقایسه‌های انجام‌شده توسط الگوریتم

باقی‌مانده یک عدد به ۲ یا ۰ است یا ۱. اگر باقی‌مانده ۰ بود عدد اول به‌عنوان برنده مقایسه در نظر گرفته می‌شود و به‌عنوان خروجی تابع مقایسه‌کننده در اختیار سایر توابع قرار می‌گیرد، درغیر این صورت، عدد دوم برگردانده خواهد شد. مقایسه‌های دقیق نیز مقایسه به‌صورت معمول انجام می‌شود.

با در نظر گرفتن این‌که اعداد ورودی دارای سه رقم اعشار هستند، زمانی که $\delta = ۰,۰۱$ باشد، حداکثر ۱۰ عدد می‌توانند در حدود δ باشند تا مقایسه نادقیق روی آن‌ها اعمال شود. بار دیگر زمانی که δ افزایش پیدا می‌کند و مقدار آن ۰,۱ می‌شود، این تعداد به ۱۰۰، افزایش می‌یابد. افزایش δ باعث می‌شود تعداد اعدادی که درحدود δ هستند، افزایش یابد. این افزایش در جهت افزایش شکل‌گیری مقایسه‌های نادقیق و احتمال افزایش خطاست. همان‌طور که در شکل (٤) مشاهده می‌کنید، در ابتدا با توجه به این‌که δ بسیار کوچک است برای تمامی مقادیر k خطا صفر است. هنگامی که δ افزایش پیدا می‌کند، در تمامی مقادیر k، این افزایش میزان خطا قابل مشاهده است. نکته جالب توجه در پراکندگی میزان افزایش خطا به ازای kهای مختلف است. به نمودار مربوط به $\delta = ۰.۱$ دقت کنید. رفتار همگن میزان افزایش خطا در kهای مختلف، نشان می‌دهد که افزایش خطا به ازای تغییر میزان k بسیار کم است. هرچقدر δ افزایش می‌یابد پراکندگی میزان افزایش خطا بیشتر می‌شود. با افزایش ۱۰۰۰ برابری δ نسبت به



شکل ۴: تاثیر پارامتر δ بر خطا یا همان K'

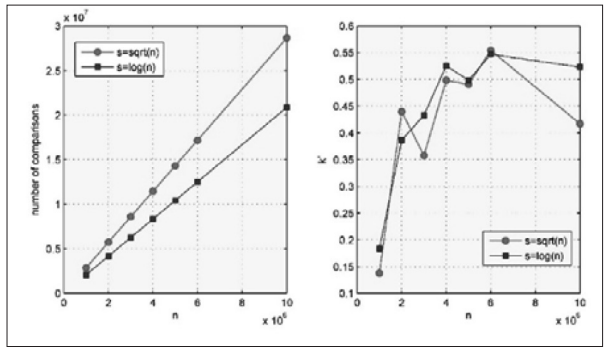
ابتدای کار، دیگر شاهد تغییر رفتار مشابه در میزان افزایش خطا به ازای مقادیر مختلف k نخواهیم بود. با دنبال کردن سیر افزایشی میزان δ، می‌توان رفتار رو به رشد میزان پراکندگی افزایش خطا را دید.

تاثیر پارامتر S

الگوریتم هر بار s عنصر را به دلخواه انتخاب می‌کند و روی آن‌ها تورنمنت با گردش نوبت برگزار می‌کند. انتخاب‌های متفاوت این s عنصر موجب نتایج متفاوت در خروجی می‌شود. پس اگر نظم یا الگوی خاصی را برای انتخاب این عناصر به الگوریتم ابلاغ کنیم، رفتار الگوریتم به ازای همان چینش خاص مشاهده می‌شود و حتما روی نتیجه خروجی نیز تاثیر گذار خواهد بود. بنابراین در پیاده‌سازی این s عنصر، به تصادف انتخاب می‌شوند.

در نتایج نظری به‌صورت پیش فرض مقدار s در الگوریتم 2-MaxFind، $s = \lfloor \sqrt{n} \rfloor$ و در الگوریتم 1-Cover مقدار $s = \lfloor 2\sqrt{n} \rfloor$ در نظر گرفته شده بود. در پیاده‌سازی، مقدار s به ترتیب برای هرکدام از توابع به $s = \lceil \log n \rceil$ و $s = \lceil 2\log n \rceil$ تغییر دادیم. نتایج این تغییرات در شکل (۵) قابل مشاهده است.

همان‌طور که انتظار داشتیم کاهش اندازه s موجب



شکل (۵): تاثیر کاهش مقدار پارامتر S روی خطا و تعداد مقایسه‌ها

کاهش تعداد مقایسه‌ها و افزایش خطا شد.

۵- نتیجه‌گیری

در این مقاله، رفتار الگوریتم محاسبهٔ k-بیشینه در مدل مقایسه نادقیق که ممکن است نتیجهٔ مقایسه داده‌های نزدیک به هم اشتباه باشد، مورد بررسی قرار گرفت. داده‌های مورد استفاده، داده‌های تولید شده تصادفی با سه توزیع مرتب‌شده، یکنواخت و خوشه‌ای است و تعداد مقایسه‌های انجام شده در اجرای الگوریتم و میزان خطای حاصل در بین نمونه‌های مختلف مقایسه گردیدند.

در مجموع، در داده‌های مرتب‌شده، تعداد مقایسه‌های انجام شده به طور ملموسی از سایر توزیع‌ها بیشتر است اما تفاوت ملموسی بین سایر توزیع‌ها در این خصوص مشاهده نشد. همچنین میزان خطا و تعداد مقایسه انجام شده در عمل، به میزان زیادی با نتایج نظری آن‌ها متفاوت است. تعداد مقایسه‌ها برای مقادیر مختلف k نیز بررسی گردید که تفاوت زیادی در تعداد مقایسه‌ها در بین مقادیر مختلف k مشاهده نگردید.

همچنین با تغییر میزان نادقیقی (پارامتر δ) نیز نشان داده شد که به ازای δهای کوچک، میزان خطا تقریباً صفر است و با افزایش میزان δ، میزان خطا افزایش پیدا می‌کند و مقدار k نیز در میزان خطا تاثیرگذار است.

به‌عنوان کارهای آتی، بررسی الگوریتم‌های پیچیده‌تر نظیر مرتب‌سازی داده‌ها در این مدل در عمل به نظر جالب توجه است.

سپاسگزاری

از بهنام ایرانفر و امیر مصری‌خانی جهت راهنمایی در فهم بهتر برخی مفاهیم نظری سپاسگزاری می‌شود.

مراجع

1. Nasibeh Siadaty؛ Karim Mohammadi, “Design and Simulation of Fault Tolerant Algorithms in Network-on-Chip”, Journal of Mathematical Modellings, Vol. 16. No. 2, pp. 33-43, 2010.
2. Yao A C., and Yao F F., “On fault-tolerant networks for sorting”, SIAM Journal on Computing, Vol. 14, pp. 120–128. January, 1985.
3. Ravikumar B., Ganesan K, and Lakshmanan K B., “On selecting the largest element in spite of erroneous information”, Annual Symposium on Theoretical Aspects of Computer Science, pp. 88–99, Springer, 1987.
4. Ajtai Miklós, Feldman Vitaly, Hassidim Avinatan and Nelson Jelani., “Sorting and selection with imprecise comparisons”, ACM Transactions on Algorithms (TALG) , Vol. 12, pp. 19, November, 2016.