نمونه سوالات مربوط به بخش1.5

1.5.1 در قطعه کد مربوط به count sort چه لزومی دارد که در حلقه for انتهایی،حلقه از n-1 شروع شود؟در حالیکه با داشتن for صعودی با شروع از 1 نیز عمل مرتب سازی انجام میشود.

راه حل:

برای **پایدار** بودن count sort باید این حلقه از n-1 شروع شده و نزولی باشد.چون اعداد در آرایه نهایی که نتیجه مرتب شده عناصر میباشد ،از انتها به ابتدای آرایه پر میشوند ،پس باید از دو عددی که یکی هستند،عدد دوم زودتر وارد آرایه شود و این به معنی شروع شدن حلقه از n-1 میباشد.در غیر اینصورت ترتیب اولیه اعداد یکسان درآرایه نهایی به هم خورده و count sort ،خاصیت پایدار بودن خود را از دست میدهد.

1.5.2 آرایه ای از 5n عدد صحیح از اعداد صحیح 1 تا n داریم.الگوریتمی با مرتبه زمانی خطی ارایه دهید که به کمک آن بتوان اعداد تکراری را حذف کرد.

راه حل:

چون تمامی اعداد صحیح بوده و در بازه 1 تا n میباشند میتوان از count sort برای مرتب کردنشان استفاده کرد.بنابراین هزینه O(5n + n) = O(n) را داده و اعداد را مرتب میکنیم.حال از یک استک برای حذف اعداد تکراری استفاده میکنیم.بدین گونه که عدد اول را در استک push میکنیم.حال برای عناصر بعدی،اگر عنصر جاری با عنصر سر استک یکی بود،سراغ عنصر بعدی می رویم و درغیر اینصورت آنرا در استک push میکنیم و همین کار را برای عناصر بعدی تکرار میکنیم.در آخر،اعداد موجود در استک،جواب مورد نظر ماست.

1.5.3 لیستی از n-1 عدد صحیح که هر عدد بین 1 تا n است،داده شده است.هیچ دو عدد برابر نیستند.الگوریم بهینه ای ارایه دهید که به کمک آن،عددی را که در این لیست نیامده است پیدا کرد.

راه حل:

چون تمامی اعداد صحیح بوده و در بازه **1** تا **n** میباشند پس با هزینه **O(n)** وبا استفاده از **count sort** میتوان این اعداد را مرتب کرد.حال از ابتدای لیست مرتب شده شروع به پیمایش میکنیم.ایندکس اولین خانه ای که در آن ایندکس مربوط به آن خانه با مقدار خانه متفاوت باشد،جواب مورد نظر است.(با فرض اینکه ایندکس خانه ها از **1** تا **n** میباشد.).پس با هزینه زمانی **O(n)** عدد موردنظر یافت میشود.

1.5.4 فرض کنید n عدد صحیح بین 1 و k دارید و میخواهید بدانید چند تا از آنها در بازه [a ,b] هسستند به طوریکه $a > 1$ و $b < k$ میباشد.الگوریتمی طراحی کنید که بعد از گرفتن زمان $\theta(n+k)$ برای پردازش کردن اعداد صحیح،بتواند پاسخ این سوال را با $\theta(1)$ بدهد.

راه حل:

ایده اصلی همان count sort است.

C[i] را تعداد اعداد صحیح ورودی کوچکتر از i تعریف میکنیم.داریم:

```
Range Preprocessing (A, n, k) begin
  for i = 0 to k do
    C[i] = 0
  for i = 1 to n do
    C[A[i]] += 1
  for i = 1 to k do
     C[i] += C[i-1]
End
```

و برای پاسخ به سوال در زمان O(1) داریم:

```
Range Query (a, b)
      return C[b] – C[a-1]
```

نمونه سوالات مربوط به بخش 2.5

2.5.1 نشان دهید چگونه می توان n عدد صحیح را که در بازه ی $[1,n^2]$ هستند، در زمان $\theta(n)$ مرتب کرد.

راه حل:

می دانیم که **radix sort** برای **n** تا عنصر که هر کدام از **d** رقم تشکیل شده اند .اگر در حلقه ی داخلی از **count sort** استفاده کنیم ($d(n+k)$) زمان می گیرد(.برای عناصر بین **[1, k]**)
حال اگر با این روش بخواهیم به **o(n)** برسیم، می بایست **d = O(n)** و **k = O(n)** باشد.
فرص کنیم هر عنصر ai را بصورت (qi,ai) بنویسیم که در آن
$qi = ai/n$ و $ri = ai \bmod n$ تعریف می کنیم.
از آنجایی که $ai \epsilon [1,n2]$ ، پس $ri \epsilon [1,n]$ و $qi \epsilon [1,n]$ ، پس **d = 2** و**k = n**
بنابر این، **radix sort** در اینجا **O(n)** زمان می برد.
_نکته :در واقع این روش می تواند **n** عدد صحیح در بازه ی **[1,O(1)]** را در مرتبه ی زمانی **O(n)** مرتب کند.

2.5.2 سریعترین الگوریتمی را ارایه دهید که به کمک آن بتوان nرشته حرفی به طول حد اکثر هشت حرف را مرتب کرد.

راه حل:

از **radix sort** برای این کار استفاده میکنیم.میدانیم که کد اسکی هر کاراکتر بین **0** تا **255** میباشد.پس از سمت راست ترین کاراکتر شورع میکنیم و در هر مرحله رشته ها را براساس آن کاراکتر مرتب میکنیم.(از **count sort** در هر مرحله استفاده میکنیم.)

چون هزینه هر **count sort** در هر مرحله**O(n+ k) = O(n +256) = O(n)** میباشد و در کل حداکثر هشت مرحله این کار تکرار میشود.**(d = 8)**.پس هزینه نهایی در کل برابر است با:**8 * O(n) = O(n)** .پس با هزینه زمانی **O(n)** و با استفاده از **radix sort** میتوان این کار را انجام داد.

2.5.3 آرایه ای از اعداد که در بازه 0 تا 1000 میباشند داده شده است.شبه کد تابعی را بنویسید که این آرایه و یک مقدار x را گرفته و زوج مرتبی از اعداد این آرایه که مجموع آن ها برابر مقدار x میباشد را پیدا کرده و باز گرداند.در صورتیکه چنین زوجی از اعداد یافت نشد، null برگرداند.هزینه زمانی تابع شما باید O(n) باشد.

راه حل:

چون اعداد در بازه **0** تا **1000** هستند پس میتوان با استفاده از **count sort** یا **radix sort** آرایه داده شده را در زمان **O(n)** مرتب کرد.سپس دو اشاره گر به ابتدا و انتهای این آرایه مرتب شده میگیریم و شروع به پیمایش آرایه مورد نظر میکنیم.اگر مجموع خانه هایی که اشاره گر ها اشاره میکنند برابر با **x** باشد که به جواب مورد نظر رسیده ایم و جواب را برمیگردانیم.اگر این مجموع کوچکتر از **x** باشد،چون آرایه مورد نظر مرتب شده است پس اشاره گر اشاره کننده به ابتدای آرایه را یکی به سمت راست آرایه حرکت میدهیم و اگر این مجموع بزرگتر باشد اشاره گر دومی را یک واحد به سمت چپ انتقال میدهیم.اگر این دو اشاره گر به هم رسیدند،چنین زوجی یافت نشده است و در آرایه موردنظر وجود ندارند.پس میتوان تابع موردنظر را بدین صورت نوشت:

```
SUM_X(S ,x)

{

    RADIX_SORT(S);

    i = 1;

    j = n;

    while i < j

    {

       do if S[i] + S[j] < x

           then  i = i+1;

       else if S[i] + S[j] > x

            then  j = j-1;

       else

              return S[i] ,S[j];

    }

if  i == j

   then return null;

}
```

2.5.4 الگوریتمی ارایه دهید که بوسیله آن بتوان یک لیست از تاریخ ها را مرتب کرد.

راه حل:

هر تاریخ از سه بخش سال و ماه و روز تشکیل شده است که هر بخش عدد صحیح در یک بازه مخصوص است.پس برای این کار سه مرحله زیر را در نظر اعمال میکنیم:

1: تاریخ ها را ابتدا بر اساس روز آنها مرتب میکنیم و برای این کار از **count sort** استفاده میکنیم.

2: سپس آن ها را بر براساس ماه آنها مرتب میکنیم و برای این کار نیز از **count sort** استفاده میکنیم.

3 : سپس آن ها را بر براساس سالشان مرتب میکنیم و برای این کار از **count sort** استفاده میکنیم.

در هر مرحله هزینه **O(n)** میدهیم و در کل سه مرحله داریم.پس هزینه نهایی برابر **O(n)** میباشد.

2.5.5 اگر در radix sort در هر مرحله به جای count sort ،از merge sort استفاده شود،آیا هزینه مرتب سازی بهتر خواهد شد؟بررسی کنید.

راه حل:

چون در **radix sort** ،در d مرحله از **count sort** استفاده میشود پس هزینه آن به صورت **O(d * (n + k))** در می آید.حال اگر **k = O(n)** باشد به صورت **O(d*n)** خواهد بود.

اگر از **merge sort** استفاده کنیم هزینه کلی به صورت **O(d * nlogn)** خواهد بود.پس اکر هزینه هر **count sort** در هر مرحله **O(n)** باشد استفاده از **count sort** بهتر است.

نمونه سوالات مربوط به بخش 3.5

3.5.1 میدانیم که در bucket sort اگرتمرکز داده ها در بازه ای بیشتر باشد،باعث میشود که الگوریتم از حالت خطی بودن خارج شود.فرض کنید که تمامی داده ها اعدادی بین 0 تا 1 میباشند.راه حلی ارایه دهید که به بتوان تا حدودی این تمرکز داده ها را از بین برده و باعث پخش شدن آنها شد.

راه حل:

میتوان از توابع ریاضی مانند **رادیکال** استفاده کرد.میدانیم که ریشه دوم اعداد بین صفر تا یک از خود اعداد بزگترند.به عنوان مثال ریشه دوم عدد 10^{-6} ،عدد 10^{-3} و ریشه دوم عدد 10^{-8}، عدد 10^{-4} میباشد که عدد بزگتریست.و نیز به عنوان مثال به وضوح داریم:

$$10^{-3} - 10^{-4} > 10^{-6} - 10^{-8}$$

یعنی توانستیم فاصله بین اعداد را بیشتر کرده و باعث پخش شدن داده ها شویم.اگر این تمرکز داده ها در بازه ای بیشتر باشد میتوانیم از ریشه های بالاتر مانند ریشه شوم و امثال آن نیز بهره ببریم.

3.5.2 اگر در bucket sort برای مرتب سازی هر bucket ،از merge sort به جای insertion sort استفاده کنیم چه تغییری در هزینه زمانی این الگوریتم حاصل میشود؟

راه حل:

بهترین حالت : n عنصر به صورت یکنواخت بین **k** سطل تقسیم شوند.

_ در این صورت هر سطل **n/k** عنصر دارد.

_ هزینه مرتب سازی هر سطل **O(n/k * log(n/k))** میباشد.

_ اگر **$k = \theta(n)$** باشد،هزینه کل برابر است با:

$K * (n/k * \log(n/k) = n\log(n/k) = O(n) \quad : k = \theta(n)$

بدترین حالت: تمامی عناصر در یک **bucket** قرار گیرند.

در این صورت هزینه کل برابر با هزینه **merge sort** خواهد بود که برابر است با **O(nlogn)**.

پس در کل میتوان گفت که در بهترین حالت باز هزینه زمانی **O(n)** پابرجاست ،ولی در بدترین حالت هزینه زمانی کمتری نسبت به زمانی که از **insertion sort** استفاده میشود،دارد.(هزینه زمانی **O(nlogn)** برای **merge sort** و هزینه زمانی **$O(n^2)$** برای **insertion sort**).

3.5.3 با استفاده از bucket sort داده های زیر را مرتب کنید.(bucket ها را در هر مرحله نشان دهید.)

165 ,659 ,425 ,159 ,893 ,544 ,785

راه حل:

Bucket 1: 165 ,159 sorted Bucket1: 159 ,165

Bucket 2: 425 sorted Bucket2: 425

Bucket 3: 569 ,544 sorted Bucket3: 544 ,569

Bucket 4: 785 sorted Bucket4: 785

Bucket 5: 893 sorted Bucket5: 893