"مید نامم آنکه جان را فکرت آموخت"

تمرین سری دوم طراحی الگوریتم

استاد: دکتر مرضیه رحیمی

دانشجو: حامد یحیی زاده مقدم

شماره دانشجویی: ۹۷۲۳۹۰۳

جواب سوال دوم) الگوریتم Heap Sort به این صورت است که ابتدا باید آرایه ی دریافتی به یک maxheap تبدیل کنیم که این کار با استفاده از تابع Build-max heap انجام می شود:

```
Build max heap (A,n){
    for(i=n/2 ; i>=1 ; i--)        => O(n)
        maxheapify(A,i,n);
    }
```

حال بعد از ساخته شدن max heap و عمل حذف را از heap انجام می دهیم، از آنجایی که عمل حذف در heap از ریشه انجام می شود پس بعد از حذف ما بزرگترین عنصر آرایه اولیه را در اختیار خواهیم داشت حال بعد از حذف باید دوباره نیاز max heap بودن به heap ما برگردانده شود که این کار با استفاده از تابع maxheapify انجام می شود:

```
max-heapify(A,i,n)
1.  l ← left(i)
2.  r ← right(i)
3.  if (≤n and A[l] > A[i]
4.  then largest ← l                    => O(h) =
5.  else larget ← i                        = O(lgn)
6.  if r≤n and A[r] > A[largest]
7.  then largest ← r
8.  if largest ≠ i
9.  then exchange A[i] ↔ A[largest]
10. max-heapify(A, largest, n)
```

9

پس کل الگوریتم heapsort به صورت زیر می‌باشد:

1. Build-maxheap (A)
2. for i ← length[A] down to 2
3. do exchange A[1] ↔ A[i]
4. max-heapify (A, 1, i-1)

$\Rightarrow O(n \lg n)$

از آنجایی که برای یک آرایه که چه به صورت Ascending و چه به صورت discending, organize شده است ما Σ step را داریم

انجام می‌دهیم پس زمان اجرای الگوریتم هم تغییری نخواهد کرد:

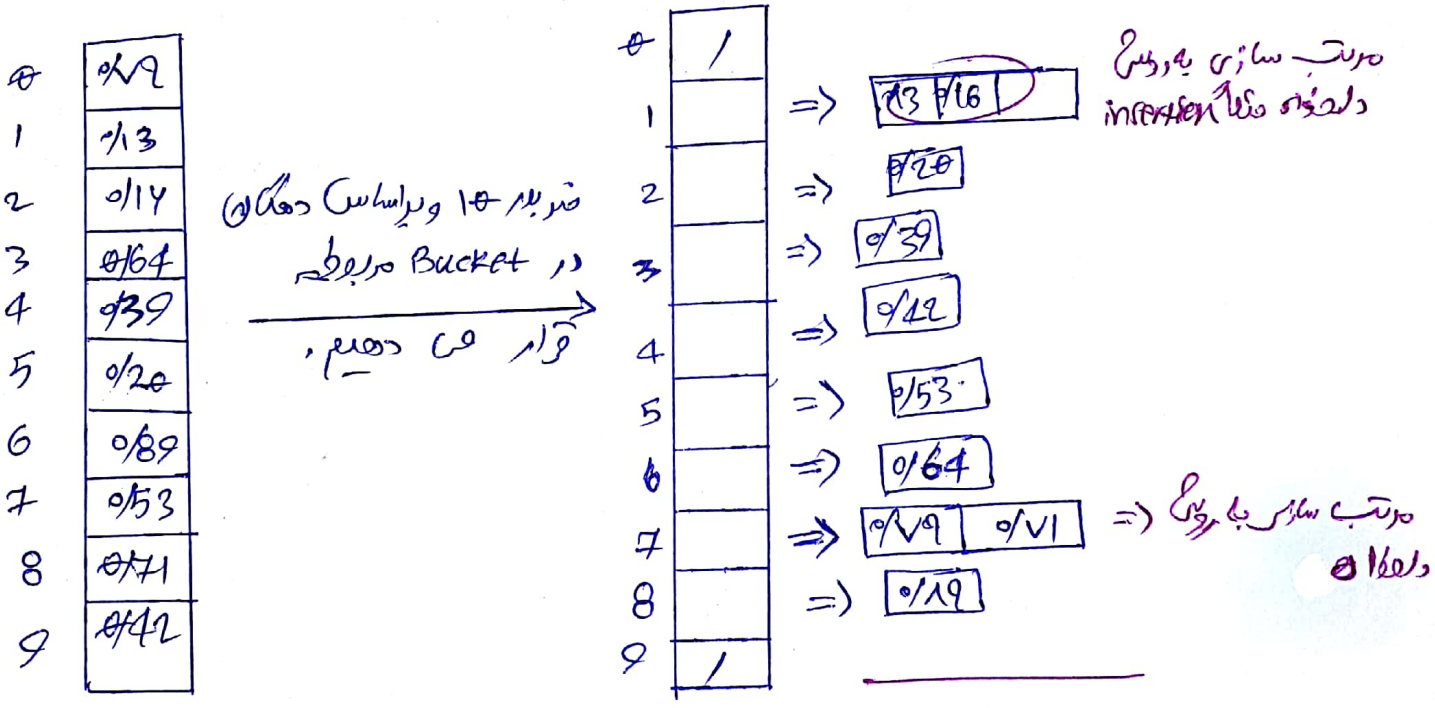
time complixity $\Rightarrow O(n \lg n)$

جواب سوال سوم)

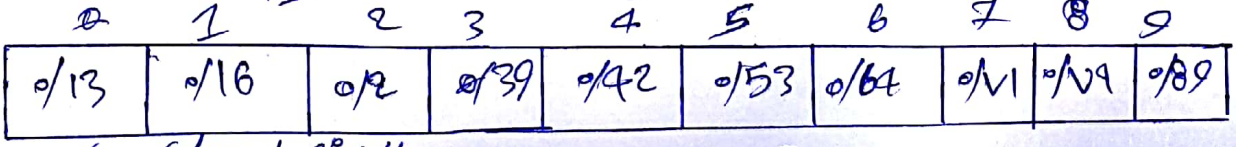
در الگوریتم Bucket Sort داده‌های ورودی خود را به پیمانه‌ها

می‌شکنیم که در هر Bucket باید خصوصیت مشترکی بین داده‌ها وجود داشته باشد مثلاً همه‌ی اعداد با 2 شروع می‌شوند. حال مطمئنیم که داده‌های موجود در هر پیمانه از پیمانه‌ی بعدی خود کوچکتر می‌باشد. حال داده‌های موجود در هر Bucket را به یک روش دلخواه sort می‌کنیم. در نهایت برای آنکه یک list مرتب شده از data های ورودی را در اختیار داشته باشیم داده‌های موجود در پیمانه‌ها را با هم concat می‌کنیم.

* در این مثال اعداد را ضربدر ۱۰ کرده و در پیمانه‌ها قرار می‌دهیم:



حال از ابتدا اعداد لیست داده‌ها را با هم concat می‌کنیم:



"our ordered list"

جواب سوال ٤)

Radix Sort ما ورودی ها را به بخش های کوچکتری می شکنیم
مثلاً برای کلمات به حرف هایش می شکنیم سپس list را based on
کم ارزش ترین bit مرتب می کنیم.

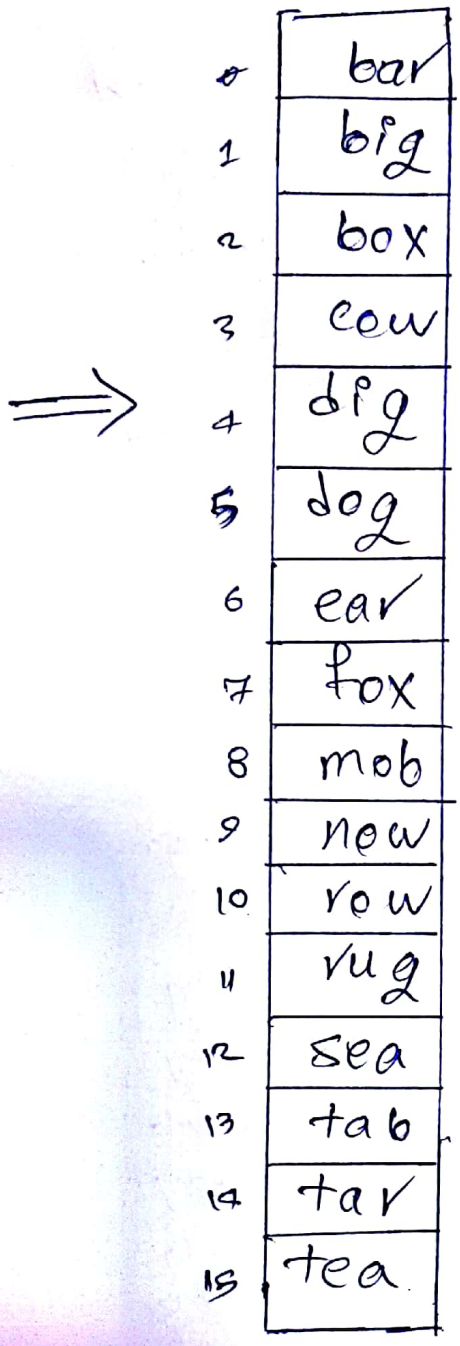
برای داده های string ⇐ بخش های کوچکتر ⇐ character

برای داده های عددی ⇐ بخش های کوچکتر ⇐ digits

	based on first Right bit ⇒	based on second Right bit ⇒
cow	sea	tab
dog	tea	bar
sea	mob	ear
rug	tab	tar
row	dog	sea
mob	rug	tea
box	dig	dig
tab	big	big
bar	bar	mob
ear	ear	dog
tar	tar	cow
dig	cow	row
big	row	now
tea	now	box
now	box	fox
fox	fox	rug

based on third Right bit ⇒

next Page



⇒ "our ordered list based on English alphabet"

جواب سوال پنجم)

الف) الگوریتمی که در زمان O(n) اجرا شود و Stable باشد
Counting Sort می‌باشد.

ب) الگوریتمی که در زمان O(n) اجرا شود و in Place باشد
Quick Sort است.

ج) الگوریتمی که Stable باشد و همچنین in Place باشد.
insertion sort می‌باشد.

د) برای ترکیب با Radix sort باید الگوریتمی Stable باشد و برای اجرای Radix در زمان O(bn) الگوریتمی باید در زمان O(n) انجام شود پس پاسخ بله است و آن الگوریتم هم الف است.
↙
Counting Sort

جواب سوال ششم

می توانیم به جای کپی کردن هر element در It's correct Position در new array ما هر element با عنصری که در correct Position خود قرار دارد swap کنیم.

حال algorithm در $O(n+k)$ اجرا می‌گردد ولی Stable نیست ولی in Place می‌باشد.

* "از آنجایی که در صورت سوال تاکیدی بر نوشتن کد نشده، کدی ننوشتم."

جواب سوال هفتم)

می توانیم از Radix sort در پایه n^2 استفاده کنیم. از آنجایی
که اعداد در مبنای n قرار دارند و دامنهٔ اعداد از 0 تا n^4-1 می باشد
بنابراین $k=n$ تعداد گذرهای مورد نیاز 3 می باشد زمانی که $n^2=100$
بنابراین $d=3$ پس داریم:

the running time of Radix Sort is:

$$O(d(n+k)) = O(3(n+k)) =$$

$$= O(3n+3n) = O(6n) \approx O(n)$$