CS5800: Algorithms — Spring '21 — Virgil Pavlu

Homework 3
Submit via Gradescope

Name:
Collaborators:

Instructions:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the grading policy outlined in the course information page.
- You must also write down with whom you worked on the assignment. If this changes from problem to problem, then you should write down this information separately with each problem.
- Problem numbers (like Exercise 3.1-1) are corresponding to CLRS 3^{rd} edition. While the 2^{nd} edition has similar problems with similar numbers, the actual exercises and their solutions are different, so make sure you are using the 3^{rd} edition.

1. ***(10 points)*** *Exercise 8.1-3.*

Solution:
Binary tree has height not more than leaves 2^h, hence
$n! \leq l \leq 2^h$

Using Sterling's Approximation,
$n! = (\sqrt{2\pi * n}).(n/e^n).(1 + \theta(1/n))$
$n! = O(n^n)$
$n! = \omega(2^n)$
$log(n!) = \theta(nlogn)$
$n \geq log(n!)$
$h = \Omega(nlogn)$
Hence $h \geq \Omega(nlogn)$ here too.

Now, we half n, so we write it as,
$h \geq log(n!/2)$
$h \geq logn! - log2$
$h \geq logn! - 1$
$h \geq nlogn - 1$
Hence $h \geq \Omega(nlogn)$ here too.

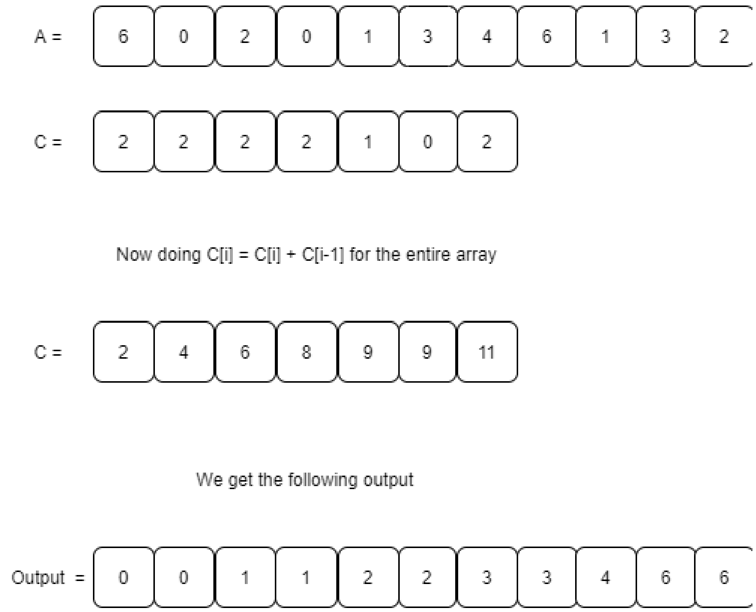
Now we divide it by 2^n
$h \geq log(n!/2^n)$
$h \geq logn! - log2^n$
$h \geq logn! - n$
$h \geq nlogn - n$
Hence $h \geq \Omega(nlogn)$ here too.

2. ***(15 points)*** *Exercise 8.1-4.*

Solution:
Every n/k sub sequence would having k elements would have $k!$ combinations. There are n/k such sequences, hence that would result to $(k!)^{n/k}$. When we arrange these in a binary tree with a 2^h height.
$2^h = (k!)^{n/k}$
$h = n/klog_2(k!)$
$h = (n/k)(k/2log_2k)$
$h = n/2(log_2k - log_22)$
$h = (nlog_2k - n)/2$
Therefore the lower bound = $\Omega(nlog_2k)$.

3. ***(5 points)*** *Exercise 8.2-1.*

Solution:
A = 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2



Range = [0:6]
Creating 7 buckets and filling each bucket with the corresponding element in the list
We groups of each element making a list = [0 0 1 1 2 2 3 3 4 6 6]

4. ***(5 points)*** *Exercise 8.2-4.*

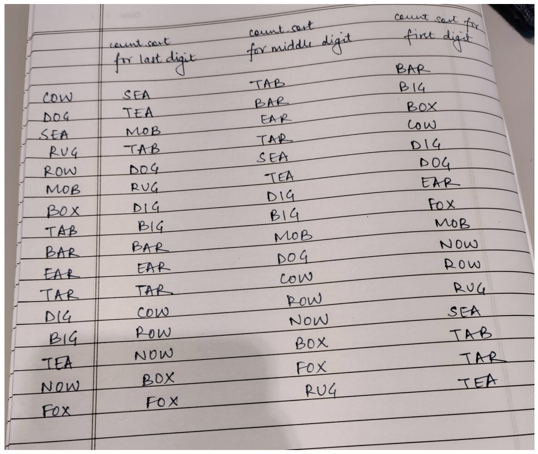
Solution:

```
COUNTING_SORT(A, B, k)
    for i = 0 to k
        C[i] = 0
    for j = 1 to A.length
        C[A[j]] = C[A[j]] + 1
        return C[b] - C[a-1]
```

`The range would be would be C[b] - C[a-1]`

5. ***(5 points)*** *Exercise 8.3-1.*

Solution:



6. ***(10 points)*** *Exercise 8.3-3.*

Solution:
If Radix sort works for k bits, then it will work for k+1 bits as well. The k terms would be sorted and when we reach the k+1th term, if it is the same we will consider the most significant digit. We assume that the sort done before this is stable. We maintain the relative order if we get one digit same as another element, which keeps it stable.

7. ***(5 points)*** *Exercise 8.3-4.*

Solution:
The time complexity for RADIX SORT is $\theta((b/r)(n+2^r))$ with $r \leq b$ and each key being [b/r]. The range is taken from 0 to $2^3 - 1$. For each pass RADIX SORT takes $\theta(n+k)$ time.

Here, k = $n^3 - 1$

Hence, we take the base as n here. For each pass each number will have $log_n(n^3)$ digits, = 3 passes. This take a time complexity of O(n) using counting sort.

8. ***(20 points)*** *Exercise 9.1-1.*

Solution:
To compare the array initially it will take n-1 comparisons. The second comparison for finding the second smallest element would take logn-1 comparisons given that we put the values in a binary tree since we have to compare again in the elements which of the values would be the second smallest. Hence adding these two we get (n-1) + (lgn-1) as the total number of comparisons, which is equal to n + lgn - 2.

9. ***(10 points)*** *Exercise 9.3-7.*

Solution:

```
1. median = QuickSelect(A, n/2)
When there are odd number of elements, median = n/2
When there are even number of elements, median = (n/2 + n+1/2)/2
OR
```

```
median = min(n/2, n+1/2)
2. Calculate absolute difference between the median and all the other elements : |a1-m|, |a2-m
3. Now add all the differences in a new array, C
4. Now from the array C, QuickSelect(C, k) find the k element
5. Now select all the elements below the kth element and return it
```

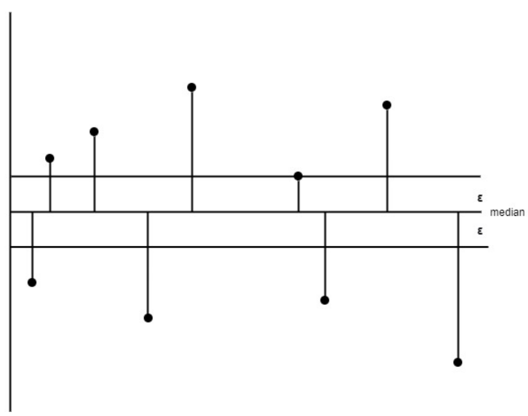
10. ***(20 points)*** *Exercise 9.3-8.*

Solution:

```
1. median = QuickSelect(X, Y, n/2)
When there are odd number of elements, median = n/2
When there are even number of elements, median = (n/2 + n+1/2)/2
OR
median = min(n/2, n+1/2)
1. if median(X) == median(Y)
    return the value
2. def median(X, Y)
    if median(X) > median(Y)
        cut X[0...n/2]
        cut Y[n/2...n]
3.  if median(X) < median(Y)
        cut Y[0...n/2]
        cut X[n/2...n]
4. call the median function recursively
5. return the median
```

11. ***(15 points)*** *Exercise 9.3-9.* ***Solution:***

We need to find $min([\sum_{n=1}^{n}(m - Yk)])$*,which is the distance between the median and the wells Yk.*



$U\epsilon - L\epsilon < 0$
$upperdistance = +\epsilon$
$lowerdistance = -\epsilon$
The line will give the most optimum results at the median, when it shifts to + epsilon and - epsilon the results will change in a negative way.

12. ***(10 points)*** *Exercise 8.4-3.*

Solution:

There are 4 cases when we flip a coin, HH, HT, TH, TT. Each of them has a probability of 1/4. For finding E[x] = $p(x)x$ which is equal to 1.

We find E[X] as following: $p(2)2 + p(1)1 + p(0)0$

E[X] = $(1/4)2 + (1/2)1 + 0$

E[X] = 1

Hence, we find $E^2[X] = (1)^2 = 1$

We find $E[X^2] = p(2)(2^2) + p(1)(1^2) + p(0)(0^2)$

$E[X^2] = (1/4)4 + (1/2)1 + 0$

$E[X^2] = 1.5$

13. ***(Extra credit 10 points)*** *Problem 9-1.*

Solution:

14. ***(Extra credit 20 points)*** *Problem 8-1.*

Solution:

15. ***(Extra credit 20 points)*** *Problem 8.4.*

Solution: