CS5800: Algorithms — Spring '21 — Virgil Pavlu

Homework 3
Submit via Gradescope

Name: Shri Datta Madhira - NUID: 001557772
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:
Inputs i = n!/2, n!/n, n!/2^n.
The best case running time for any comparison sort is $\Omega(n \lg n)$.
For i = n!/2, we get,

$$\lg(n!/2) = \lg n! - \lg 2 = n \lg n - 1 = \Omega(n \lg n)$$

For i = n!/n, we get,

$$\lg(n!/n) = \lg n! - \lg n = n \lg n - 1 = \Omega(n \lg n)$$

For i = n!/2^n, we get,

$$\lg(n!/2^n) = \lg n! - \lg 2^n = \lg n! - n = \Omega(n \lg n)$$

As we can see, there is no comparison sort whose running time is linear for at least half of the n! inputs of length n.

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

Solution:
For each sequence of k elements, we have k! leaves.
For $\frac{n}{k}$ sequences, we get $(k!)^{(\frac{n}{k})}$ leaves.
From Theorem 8.1 we can say that

$$(k!)^{(\frac{n}{k})} \leq 2^h$$

Applying lg on both sides,

$$(\frac{n}{k}) \lg k! \leq h$$

$$(\frac{n}{k}) k \lg k \leq h$$

$$n \lg k \leq h$$

$$h = \Omega(n \lg k)$$

Hence, we have proved that lower bound on the number of comparisons needed to solve this variant of the sorting problem.

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

Solution:
The given array A = $< 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2 >$.
Let the k = 7, which implies that array C will be C[0 $\cdots$ k].
Running the counting sort **(the version taught in class)** for each element in A, we get,

$$
\begin{aligned}
A[i] = 6 : C = < 0, 0, 0, 0, 0, 0, 1, 0 > \\
A[i] = 0 : C = < 1, 0, 0, 0, 0, 0, 1, 0 > \\
A[i] = 2 : C = < 1, 0, 1, 0, 0, 0, 1, 0 > \\
A[i] = 0 : C = < 2, 0, 1, 0, 0, 0, 1, 0 > \\
A[i] = 1 : C = < 2, 1, 1, 0, 0, 0, 1, 0 > \\
A[i] = 3 : C = < 2, 1, 1, 1, 0, 0, 1, 0 > \\
A[i] = 4 : C = < 2, 1, 1, 1, 1, 0, 1, 0 > \\
A[i] = 6 : C = < 2, 1, 1, 1, 1, 0, 2, 0 > \\
A[i] = 1 : C = < 2, 2, 1, 1, 1, 0, 2, 0 > \\
A[i] = 3 : C = < 2, 2, 1, 2, 1, 0, 2, 0 > \\
A[i] = 2 : C = < 2, 2, 2, 2, 1, 0, 2, 0 >
\end{aligned}
$$

The sorted version of array A is, A = $< 0, 0, 1, 1, 2, 2, 3, 3, 4, 6, 6 >$.

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

Solution:

Algorithm 1: Algorithm to find how many of the ’n’ integers fall into a range [a$\cdots$b] - Using Counting Sort

```
Function CountingSort(A, B, k, a, b):
    For i = 0 to k
        C[i] = 0
    For j = 1 to A.length
        C[A[j]] += 1
    For i = 1 to k
        C[i] = C[i] + C[i-1]
    If a>0 :
        Return C[b] - C[a - 1]
    Else
        Return C[b]
```

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

Solution:

COW	SE**A**	B**A**R	**B**AR
DOG	TE**A**	E**A**R	**B**IG
SEA	TA**B**	T**A**B	**B**OX
RUG	MO**B**	T**A**R	**C**OW
ROW	BI**G**	S**E**A	**D**IG
MOB	DI**G**	T**E**A	**D**OG
BOX	DO**G**	B**I**G	**E**AR
TAB	RU**G**	D**I**G	**F**OX
BAR	BA**R**	B**O**X	**M**OB
EAR	EA**R**	C**O**W	**N**OW
TAR	TA**R**	D**O**G	**R**OW
DIG	CO**W**	F**O**X	**R**UG
BIG	NO**W**	M**O**B	**S**EA
TEA	RO**W**	N**O**W	**T**AB
NOW	BO**X**	R**O**W	**T**AR
FOX	FO**X**	R**U**G	**T**EA

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

Solution:
Let the number of digits be ′d′.
Assuming the Radix sort worked for d - 1 digits, i.e., d-1 digits are sorted. We have to prove it works for d-digit to prove that Radix sort works in general.
For any two numbers with d-digits, we have three options while sorting them. This applies for the d-digit too. So, for two numbers ′x′ and ′y′, we have,

```
SITUATION #1: x's d-digit > y's d-digit. In which case, we have to swap x and y.

SITUATION #2: x's d-digit < y's d-digit. In which case, they remain in the same
              place. No operation needed to be performed.

SITUATION #3: x's d-digit = y's d-digit. In which case, we do not change anything
              because the numbers are already sorted w.r.to the next d - 1 digits.
              Here, we assume the sorting algorithm is stable and will not randomly
              swap the two numbers.
```

Therefore, we have proved that the Radix sort works for all the digits in the given number and sorts all the given elements in the correct order.

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

Solution:
Digit count of N = upper bound of log(N).
So, if we apply this logic, we will get the maximum number of digits for the given range as 3.
As all the numbers are integers, we know that k = 10, since each digit can take [0-9] as possible values.
So, using Radix Sort, which takes $\Theta(d(n+k))$ time to run, we get,

$$\Theta(3*(n+10)) \Rightarrow \Theta(n)$$

.

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

Solution:
For this problem, following the hint, we will first find the smallest element of the list. It will take n - 1 comparisons, because we compare the element to every element other than itself. The height of the comparison tree for this will be $\lg n$.
So, for the second smallest element we need to traverse the tree until the penultimate element which will be our second smallest element. This takes $\lg n - 1$ time.
So, in total we take, (n - 1) for finding the smallest element and $(\lg n - 1)$ time for finding the second smallest element.
Therefore, the final running time will be,

$$(n-1)+(\lg n-1) = n - \lg n - 2$$

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

Solution:
Algorithm 2: Algorithm to find k numbers in S that are closest to the median of S

```
Function FIND-CLOSEST(S, k):
    median = (S.length + 1)/2
    For i = 1 to S.length
        absValues[i] = abs(S[i] - median)
    e = SELECT(absValues, k)
    For i = 1 to S.length
        If absValues[i] ≤ e and i != median :
            resArray.append(S[i])
    Return resArray
```

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

Solution:

Algorithm 3: Algorithm to find median of all 2n elements in arrays X and Y

```
Function MEDIAN-OF-TWO-ARRAYS(X, Y):
    n = X.length or Y.length
    While X[i] > Y[j] and i > -1 and j < Y.length+1 :
        swap(X[i], Y[j])
        i += 1, j += 1
    Sort(X), Sort(Y)
    Return floor((X[X.length] + Y[0])/2)
```

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

Solution:

Assuming the main spur line will be the main x-axis in x-y co-ordinate system, we will calculate the distances from each well to the main line.
We will then calculate the average distance (median) and will re-arrange the main spur line to be in that distance, so that we will get the most optimal position for the line to be in from all the wells.

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

Solution:

X = Number of heads in two flips of a fair coin.
So, the possibilities of X are,

```
X = 2 times, with P(X) = 1/4
  = 1 time, with P(X) = 1/4
  = 0 times, with P(X) = 1/4
```

$$E[X^2] = 2^2.\frac{1}{4} + 2.1^2.\frac{1}{4} + 0^2.\frac{1}{4} = 4.\frac{1}{4} + \frac{1}{2}$$

$$\boldsymbol{E[X^2] = 1.50}$$

$$E^2[X] = E[X].E[X] = (2.\frac{1}{4} + 2.1.\frac{1}{4} + 0.\frac{1}{4}).(2.\frac{1}{4} + 2.1.\frac{1}{4} + 0.\frac{1}{4})$$

$$E^2[X] = (\frac{1}{2} + \frac{1}{2}).(\frac{1}{2} + \frac{1}{2})$$

$$\boldsymbol{E^2[X] = 1}$$

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:

(a)

Algorithm 4: Algorithm using $\Theta(n^2)$ comparisons to group the jugs into pairs.

```
Function WATER-JUG-A($n$):
    For $i = 1$ to $n$
        For $j = 1$ to $n$
            If $quantity[i] = quantity[j]$ :
                jug-pairs.append(i,j)
```

(b) This is a normal comparison problem albeit with a bit of a backstory. So, the proof discussed in Theorem 8.1. is the basis.
The number of comparisons will be 2n! and assuming 'h' is the height of the tree, we get,

$$2^h \leq 2n! \Rightarrow h \leq \lg 2n! \Rightarrow h = \Omega(n \lg n)$$

(c) Assume T(n) = $cn \lg n$

$$\lg n! \leq cn \lg n \Rightarrow \lg n! \leq \lg n^{cn} \Rightarrow n! < n^n$$

Therefore, it is O($n \lg n$). Hence Proved.