CS5800: Algorithms — Virgil Pavlu

Homework 3

Name: MUMUKSHA PANT
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:
let us take height h and elements n ,
Using Theorem 8.1 : Any comparison sort algorithm requires $\Omega(nlgn)$ comparisons in the worst case
1. half of n
We know , $n! \leq 2^h$
$\frac{n!}{2} \leq n! \leq 2^h$
Taking log both sides :
$\log \frac{n!}{2} \leq h$
$h \geq \log \frac{n!}{2}$
$h \geq \Omega(n \log n)$

2. 1/n of the input
$\frac{1}{n} n! \leq n! \leq 2^h$
Taking log both sides :
$\log \frac{n!}{n} \leq h \log 2$
$\log \frac{n!}{n} \leq h$
$\log n! - \log n \leq h$ Using $logn! = \theta(nlogn)$
$\theta(n \log n) - logn \leq h$

3. $1/2^n$ input :
$\frac{1}{2^n} n! \leq n! \leq 2^h$
Taking log both sides :
$\log \frac{n!}{2^n} \leq h \log 2$
$\log n! - \log 2^n \leq h$ Using $logn! = \theta(nlogn)$
$\theta(n \log n) - n \leq h$

From all the above proofs, it is confirmed that there is no comparison sort whose running time is LINEAR

---

2. (15 points) Exercise 8.1-4.

Solution:
using decision tree model , we know , each permutation appears as a rechable leaf $n! \leq 2^h$
we have [n/k] independent sorting problems each of size k .
So , $k! * k! * k!$... n/k times
$k!^{\frac{n}{k}} \leq 2^h$
taking log -
$\frac{n}{k} \log k! \leq h * log2$
using log n! = theta (n logn)
$\Omega(n * logn) \leq \frac{h*k}{n}$

---

3. (5 points) Exercise 8.2-1.

Solution:
COUNTING SORT (img attached)

Input array → (A) 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2

Range (k) = 0–6

Step 1 – Initialise Count array to 0.

for i = 0 to k

C

0	0	0	0	0	0	0

Step 2 – Count frequency of occurence of each digit in Array A & add it in C.

0	1	2	3	4	5	6
2	2	2	2	1	0	2

for i = 1 to A.length

C[A[i]] = C[A[i]] + 1

Step 3 – cumulative sum.

0	1	2	3	4	5	6
2	4	6	8	9	9	11

Step 4 – create an op array B equal to Input Array length & fill in the nos. using

0	1	2	3	4	5	6	7	8	9	10
0	0	1	1	2	2	3	3	4	6	6

OUTPUT ARRAY :

for i = A.length to 1

B[C[A[i]]] = A[i]

C[A[i]] = C[A[i]] – 1

---

4. (5 points) Exercise 8.2-4.

Solution:
Using Algorithm similar to Counting Sort :
FUNC CountOcc (A , k , a, b

1. **for i=1 to k**
   C[min:max] =0 - - - initialise count to 0
end for

2. **for** i=1 to A.length
C[A[i]] = C[A[i]] + 1 - - - find count of all distinct elements
end for

3. **return** C[b] - C[a-1]
end

---

5. (5 points) Exercise 8.3-1.

Solution:
using radix sort :
1st iteration on the least significant digit :
COW
DOG
SEA
RUG
ROW
MOB
BOX
TAB
BAR
EAR
TAR
DIG
BIG
TEA
NOW
FOX

2nd iteration :
SEA
TEA
MOB
TAB

DOG
RUG
DIG
BIG
BAR
EAR
TAR
COW
ROW
NOW
BOX
FOX

3RD iteration :

TAB
BAR
EAR
TAR
SEA
TEA
DIG
BIG
MOB
DOG
COW
ROW
NOW
BOX
FOX
RUG

FINAL SOLUTION :

BAR
BIG
BOX
COW
DIG
DOG
FAR
FOX
MOB
NOW
ROW
RUG

SEA
TAB
TAR
TEA

---

6. (10 points) Exercise 8.3-3.

Solution:
We take a number x and y , say x= 47682 AND y= 38769 of d digits
then , x1 = 2 , x2=8 , x3= 6 , x4=7 and so on..
For d=1 (Least Signifcant digit)
if x1 ¡ y1 , x comes appear before y .
if x1 = y1, x comes before y , their order will not change (assuming a stable sort was used) .

Induction hypothesis : lets assume induction is correct after $i-1 < d$ pass ,

CASE 1: $xi < yi$
x comes appear before y
CASE 2: $xi = yi$
using induction ,
x and y remain in same order that they appear before the ith iteration

---

7. (5 points) Exercise 8.3-4.

Solution:
Show how to sort n integers in the range 0 to $n^3 - 1$ in O(n) time.
Using LEMMA 8.4 from CLRS , RT = $\theta(\frac{b}{r})(n+2^r)$ if stable sort uses $\theta(n+k)$ time for input range 0 to k
we know , formula for d (no of digits)= $\log_b k$
we have k =n^3
we are ignoring -1 because we are considering asymptotic bounds
d= $\log_b n^3$
considering base to be n

d= $3.\log_n n$
=> $d = 3$
Running Time = $\theta(\frac{b}{r})(n+2^r)$

$= \theta(d*(n+n)$
$= \theta(3*(2n)$
$= \theta(6n)$
$\theta(n)$

---

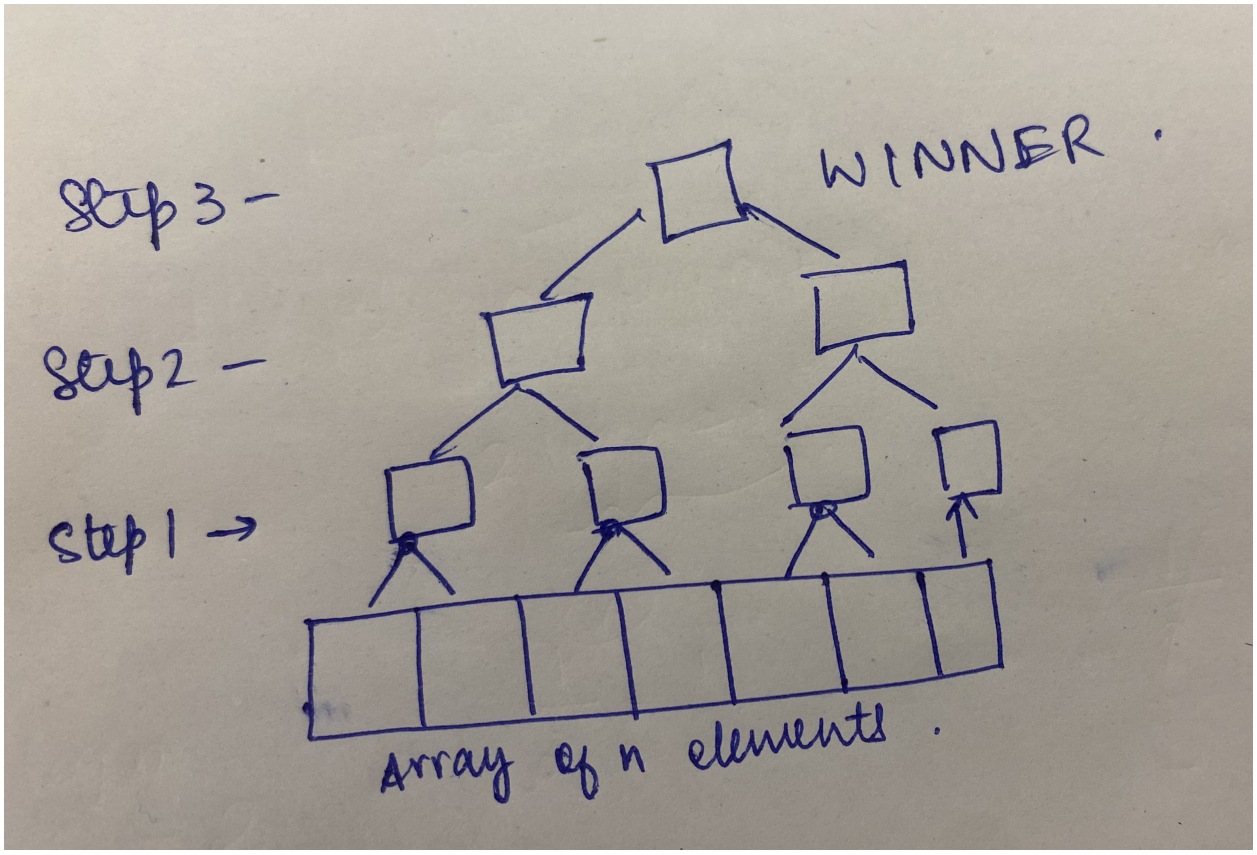
8. (20 points) Exercise 9.1-1.

Solution:

To find the MINIMUM of any Array A :

```
MINIMUM (A)
1. min= A[1]
2. FOR i=2 to n
3.     if (min > A[i])
4.         min=A[i]
5. return min
```

We iterate through each element in the Array and keep a track of the smallest element seen so far. We can consider this as a **Tournament Problem among the elements** where each of the smaller element wins . Every element **except the minimum in the Array (winner)** loses once.
We require **(n-1)** comparisons .
To find the second smallest (second minimum), we have to go to one level before we find the winner . **Because if there were no second minimum, we would have 2 winners**



Consider this as a decision tree.
We pair up the elements of an array and find the smallest among them .
In each iteration , we are doing $\frac{n}{2}$ comparisons.
In each iteration , we are returning +1 element.
Half elements from the previous level will move to next level that is : T(n/2) elements .

We require **(n-1)** comparisons .

To find the second smallest (second minimum), we have to go to one level before we find the winner .
for finding the second smallest would be height of the tree : $\log n - 1$
Running Time : $(n-1) + \log n - 1 = n + \log n - 2$

---

9. (10 points) Exercise 9.3-7.

Solution:
We will usequick select algorithms to solve this problem.
After finding the median , we iterate the array and subtract the difference of all elements from the median (find the absolute difference) .
We will get the k closest neighbour with the minimum difference

function QUICKSELECT( ) STEPS 1-15
Divide the input array into $[n/5]$ groups
1. **for** i =0 to A.length
2. subarr = [A[i : i + 5]]
end for

Run insertion sort (A) Steps 3-10
3. **for** i=2 to A.length
4. key=A[i]
5. Insert A[i] into sorted space A[1..i-1]
6. j= i-1
7. while $j > 0 and A[j] > key$
8. A[j + 1] = A[j]
9. j = j - 1
10. A[j + 1] = key
end for

Find median from sorted groups
12. median = QUICKSELECT (A[0..n-1] , n/2)

13. position = PARTITION (A , n , median)
14. $if position > k -> return QUICKSELECT(A[0..p-1], k)$
15. $if position < k -> return QUICKSELECT(A[p+1], k - position)$

end function

function kthELEMENT (A , k) Now iterate the sorted array to find the absolute difference
16. Array diff[0..A.length]
17. mainMedian = QUICKSELECT (A , n/2)
18. **for** 0 to n-1

19. diff[i]=absolute (A[i]-median)
end for

20. kthElement= QUICKSELECT (A, n/2)
21. **for** i=0 to n-1
23. **if** diff[i] ¡ kth element 24. return (diff[i] + median)
end if
end for

end

---

10. (20 points) Exercise 9.3-8.

Solution:
Collaborator : Geeks for Geeks website and Dhruv Saini

We find the median of array X and array Y independently .
Then compare the medians

ALGORITHM :

1. **func** mainMedian ( )
2. mX = median(X, startX, endX)
3. mY =median(Y, startY, endY)

4. $if(mX == mY)$
5. return mX
6. $if(mX > mY)$
7. return min(mX,mY)
8. $if(mX < mY)$
9. return min(m1,m2)

10. **func** median (arr , start , end)
11. $n = start + end/2$
12. $if(n \mod 2 == 0)$
13. return (arr[start + (n / 2)] + arr[start + (n / 2 - 1)]) / 2;
14. else return a [start+(n/2)]

end

---

11. (15 points) Exercise 9.3-9.

Solution:
collaborators : discussed during OH:
in this problem , we need to optimise the spur length from wells to the main pipeline

x coordinate can be ignored because on the x axis the main pipeline lies and therefore we dont need to optimise it.
we can only optimise the y axis.
For finding the optimum spur length, we need to find the median of y coordinate.
There will be 2 cases in this - median could be even or odd. if median is odd , we can take the y to be equal to to all the medians of the pipelines

For the second case when median is EVEN , means there are n+1/2 wells above the median and n-1/2 below the median. we get 2 medians - upper median and lower median .

We will use Quickselect algorithm to find the median in linear time

QUICKSELECT ( )
1. divide the input array into [n/5] groups
2. Run insertion sort to sort the elements
3. find the median of [n/5] elements from the sorted groups
4. now call quickselect to find the MEDIAN of all the groups.
5. Using middle elements as a pivot, we call PARTITION recursively.
6. Call QUICKSELECT recursively

---

12. (10 points) Exercise 8.4-3.

Solution:
Let x be the Heads when 2 coins are flipped = TT , TH , HT , HH
x =[0, 1, 1, 2]
x^2 = [0, 1, 1, 4]

Expectation of x = mean of x
E(x) = mean (x)
$= \frac{0+1+1+2}{4}$

$= \boxed{1}$

similarly ,

Expectation of x^2 = mean of x^2
E(x^2) = mean (x^2)

$= \frac{0+1+1+4}{4}$

$= \boxed{1.5}$

---

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

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

15. (Extra credit 20 points) Problem 8.4.