Algorithms (CSO 211) - Mid-Sem Exam

Name:- Shruti T. Avhad Roll no.:- 20074030 Branch:- CSE (IDD)

Q1.

a)

Ans- When we implement quicksort with a random pivot element, chances of having a worst case are quite low i.e picking the smallest or largest element as pivot. But generation of random number is slow.

Choosing a fixed pivot like a median is faster than random number, but then the chances of getting into a worst case is higher.

So in either case, we cannot decide if random pivoting performs better or vice-versa.

b) Assuming the set is sorted, or else sort it and then apply search.

The pseudocode for the algorithm would be:

```
Initialise low=0, high = size - 1
Repeat until low > high
        oneThird = (2*low + high) / 3
        If a[oneThird] == x
            point 'Element found' at index onethird
            break.
        Else
            If a[oneThird] < x
                Low = oneThird + 1
            Else
                high = oneThird - 1
If low > high
    print 'Element no found'
```

Recurrence relation (for average case).

$$T(n) = \frac{1}{3} \cdot T\left(\frac{n}{3}\right) + \frac{2T}{3}\left(\frac{2n}{3}\right) + O(1)$$

For worst case.

$$T(n) = T\left(\frac{2n}{3}\right) + O(1)$$

Solving the recurrence relation using method of substitution,

Let $T(n) = \Theta(\log n)$.

we need to prove that,

$$c_1 \log(n) \leq T(n) \leq c_2 \log(n) \quad \text{for some } c_1 > 0,\ c_2 > 0,\ n \geq n_0$$

Let $n \geq n_0$, where $n_0 = \frac{n}{3}$, the above relation holds,

$$c_1 \log\frac{n}{3} \leq T\left(\frac{n}{3}\right) \leq c_2 \log\frac{n}{3} \quad \text{and} \quad c_1 \log\frac{2n}{3} \leq T\left(\frac{2n}{3}\right) \leq c_2 \log\frac{2n}{3}$$

$$\frac{1}{3} c_1 \log\frac{n}{3} \leq \frac{1}{3} \cdot T\left(\frac{n}{3}\right) \leq \frac{1}{3} c_2 \log\left(\frac{n}{3}\right) \quad \& \quad \frac{2}{3} c_1 \log\frac{2n}{3} \leq \frac{2}{3} T\left(\frac{2n}{3}\right) \leq \frac{2}{3} c_2 \log\frac{2n}{3}$$

Let $O(1) = k > 0$;

$$\Rightarrow \frac{1}{3} c_1 \log\frac{n}{3} + \frac{2}{3} c_1 \log\frac{n}{3} + k \leq \frac{1}{3} T\left(\frac{n}{3}\right) + \frac{2}{3} T\left(\frac{2n}{3}\right) + k \leq \frac{1}{3} c_2 \log\frac{n}{3} + \frac{2}{3} c_2 \log\frac{2n}{3} + k$$

$$\Rightarrow c_1 \log\frac{n}{3} + \frac{2}{3} c_1 \log\frac{2}{3} + k \leq T(n) \leq c_2 \log\frac{n}{3} + \frac{2}{3} c_2 \log\frac{2}{3} + k$$

$$c_1 \log n + \left(k + \frac{2}{3} c_1 \log\frac{2}{3} - c_1 \log 3\right) \leq T(n) \leq c_2 \log n + \left(k + \frac{2}{3} c_2 \log\frac{2}{3} - c_2 \log 3\right)$$

$$c_2 \log n \in T(n) \leq c_2 \log n$$

for $k + \frac{2}{3} c_1 \log\frac{2}{3} - c_1 \log 3 \geq 0$

$$0 < c_1 \leq \frac{k}{\log 3 - \frac{2}{3}\log\frac{2}{3}}$$

for $k + \frac{2}{3} c_2 \log\frac{2}{3} - c_2 \log 3 \leq 0$

$$c_2 \geq \frac{k}{\log 3 - \frac{2}{3}\log\frac{2}{3}} > 0$$

∴ we have $c_1 \log n \in T(n) \leq c_2 \log n$. ✓ Proved.

$$\therefore T(n) = \Theta \log(n)$$

for worst case,

$$T(n) = T\left(\frac{2n}{3}\right) + O(1)$$

By master's Theorem, $\alpha = 1$, $b = 3/2$

$$O = \log_{3/2} 1$$

$O(1) = O(n^0) \Rightarrow c = 0$.

$$\therefore T(n) = O(\log n)$$

Q2.)

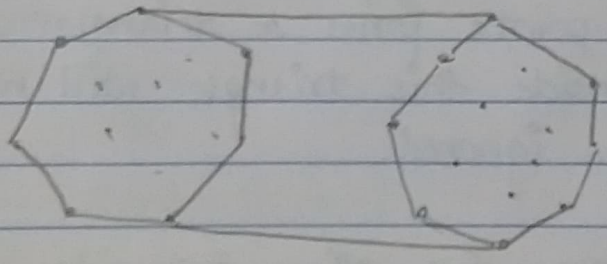
a) Convex Hull using Divide and Conquer Algorithm.

Method-1

1. From the given set of points for which we have to find the convex hull
2. Sorting all the points by their x-coordinate. $\Theta(n \log n)$ time.
3. Then,

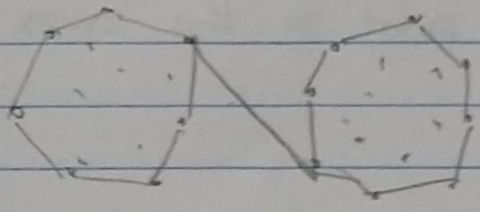
   Divide and Conquer:
   - Find the convex hull of the left half of points.
   - Find the convex hull of the right half of points.
   - Merge the two hulls into one.

4. Merging two hulls is done by finding the upper and lower tangents of the two hulls.



- Then removing points that are cut off.

5. Now rightmost point of the hull, and leftmost point of rt. hull.



∵ line is not upper tangent to both left and right.
- if line is not upper tangent to the left, move to next pt.
- while line is not upper tangent to the right, move to next pt.

6. We need to maintain them in Clock-wise or C-Clock-wise.
7. Base case for the recursion:
   - we will create a different code to find hulls of 3 (let) pts.

Method-2
Quickhull Algorithm

1. Input an array of points, with minimum x-coordinate and point with max x-coordinate.
2. Making a line joining these two points that divides the whole set into two parts.
3. Find the point on one side of the line with max. dist. from the line, this points form a triangle with line. All the points lying inside this triangle will not be part of the hull and can be ignored.
4. Repeating the prev. step for other two sides of triangle.
5. The above steps divides the problem into two sub-problems. Repeat above steps till no point is left with the hull. The end points of this point to the convex hull.

Q2.)

b) (i)

$$T(n) = n.T(n-1)+n \quad \text{for } n \geq 2, \quad T(1)=1.$$

$$T(n) = n[(n-1)T(n-2)+n-1]+n$$

$$\Rightarrow n(n-1)T(n-2) + n(n-1) + n$$

$$\Rightarrow n(n-1)(n-2)T(n-3) + n(n-1)(n-2) + n(n-1) + n$$

$$\Rightarrow \frac{n!}{1!} + \frac{n!}{2!} + \cdots + \frac{n!}{(n-1)!}$$

$$\Rightarrow n! \left(\frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{(n-1)!} \right)$$

$$\boxed{T(n) = n! \sum_{r=0}^{n-1} \left(\frac{1}{r!} \right)}$$

(ii)

$$T(n) = 2T(n-1) + n^2 - 2n + 1 = 2T(n-1) + (n-1)^2$$

$$= 2(2T(n-2) + (n-2)^2) + (n-1)^2$$

$$= 2^2 T(n-2) + 2(n-2)^2 + (n-1)^2$$

$$= 2^3 T(n-3) + 2^2(n-3)^2 + 2(n-2)^2 + (n-1)^2$$

$$T(n) = 2^{n-1} T(1) + 2^{n-2}(1)^2 + 2^{n-3}(2)^2 + \cdots + (n-1)^2$$

$$T(n) = 2^{n-1} + \sum_{k=1}^{n-1} 2^{n-1} \cdot \frac{k^2}{2^k}$$

$$T(n) = 2^{n-1} + 2^{n-1} \sum_{k=1}^{n-1} \frac{k^2}{2^k}$$

After solving the above summation series using AP·GP multiplication,

$$\sum_{k=1}^{n-1} 2^n \frac{k^2}{2^k} = -n^2 - 2n + 3 \cdot 2^n - 3.$$

$$T(n) = 2^{n-1} + \sum_{k=1}^{n-1} 2^{n-1} \frac{k^2}{2^k}$$

$$T(n) = 2^{n-1} - n^2 - 2n + 3 \cdot 2^n - 3 = \boxed{7 \cdot 2^{n-1} - n^2 - 2n - 3}$$

(iii)

$$T(n) = 4\,T\left(\frac{n}{3}\right) + 2n - 1 \qquad n = 3^k \quad k > 0, \; k \in \mathbb{N}$$

$$T(1) = 3$$

$$T(n) = 2n-1 + 4\left(4T\left(\frac{n}{9}\right) + \frac{2n}{3} - 1\right)$$

$$= 2n\left(1 + \frac{4}{3}\right) - 1\left(1+4\right) + 16\,T\left(\frac{n}{9}\right)$$

$$\Rightarrow T(n) = 2n\left(1 + \frac{4}{3} + \left(\frac{4}{3}\right)^2 + \cdots + \left(\frac{4}{3}\right)^{r-1}\right) - 1\left(1 + 4 + 4^2 + \cdots + 4^{n-1}\right) + 4^r\,T\left(\frac{n}{3^r}\right)$$

Let $r = k$.

$$\Rightarrow T(n) = 2n\left(\frac{\left(\frac{4}{3}\right)^k - 1}{\frac{4}{3} - 1}\right) - 1\left(\frac{4^k - 1}{4-1}\right) + 4^k\,T(1)$$

$$n = 3^k \qquad k = \log_3 n \qquad 4^k = 4^{\log_3 n} = \left(3^{\log_3 4}\right)^{\log_3 n} = n^{\log_3 4}$$

$$T(n) = 6n\left(\frac{n^{\log_3 4}}{n} - 1\right) - \frac{1}{3}\left(n^{\log_3 4} - 1\right) + 3n^{\log_3 4}$$

$$\boxed{T(n) = \frac{\left(26 \cdot n^{\log_3 4} + 1\right)}{3} - 6n}$$

(Q3.)

a)

Ans- **MASTER'S THEOREM**

Consider the recurrence, $T(n) = aT(n/b) + f(n)$. -(i)

where, a, b are constants.

Then,

1) If $f(n) = O\left(n^{\log_b^a - \varepsilon}\right)$ for some const. $\varepsilon > 0$, then $T(n) = O(n^{\log_b^a})$

2) If $f(n) = \Theta(n^{\log_b^a})$, then $T(n) = \Theta(n^{\log_b^a} \log n)$.

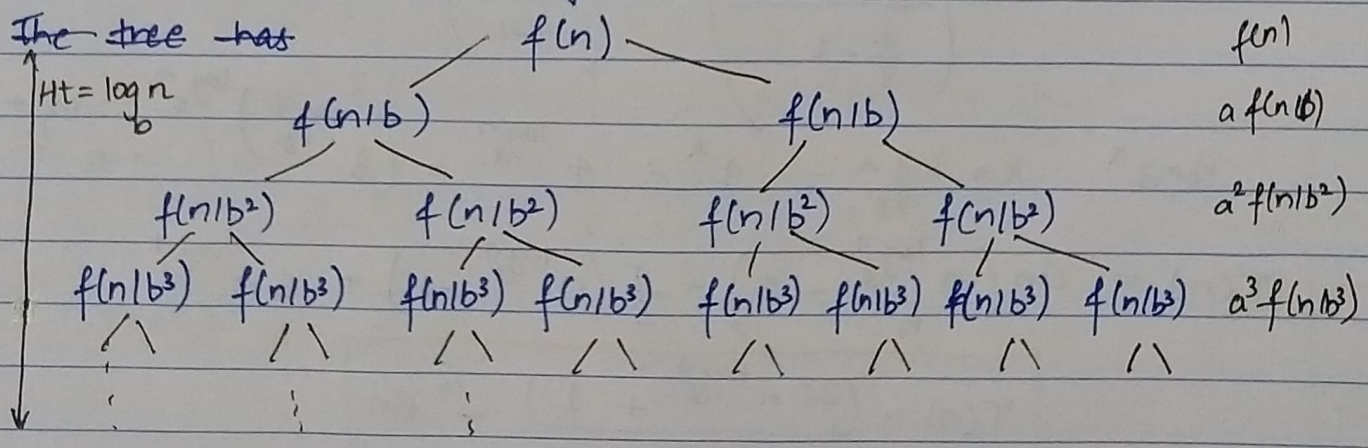
3) If $f(n) = \Omega(n^{\log_b^{a+\varepsilon}})$ for some const. $\varepsilon > 0$, and if f satisfies the smoothness condition $af(n/b) \leq cf(n)$ for some const. $c < 1$, then $T(n) = \Theta(f(n))$

PROOF:

The solution of the recurrence will be,

$$T(n) = \sum_{i=0}^{\log_b^n} a^i f(n/b^i) + O(n^{\log_b^a}) \qquad -(ii)$$

As we can see the, tree generated from recurrence (i).



The tree has depth $\log_b n$ and branching factor a.
There are a^i nodes at level i, each labeled $f(n/b^i)$.
The value of T(n) is the sum of the labels of all the nodes of the tree.
The sum (ii) is obtained by summing the i^{th} level sums.
The last term $O(n^{\log_b a})$ is the sum across the leaves, which $a^{\log_b n} f(1) = n^{\log_b a} f(1)$.
And, here $a = 2$.

CASE (1)

$$T(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i) + O(n^{\log_b a}) \leq \sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a - \varepsilon} + O(n^{\log_b a}) \quad -(iii)$$

and

$$\sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a - \varepsilon} = n^{\log_b a - \varepsilon} \sum_{i=0}^{\log_b n} a^i b^{-i \log_b a} b^{i\varepsilon} = n^{\log_b a - \varepsilon} \sum_{i=0}^{\log_b n} a^i a^{-i} b^{i\varepsilon}$$

$$= n^{\log_b a - \varepsilon} \sum_{i=0}^{\log_b n} b^{\varepsilon i} = n^{\log_b a - \varepsilon} \frac{b^{\varepsilon(\log_b n + 1)} - 1}{b^\varepsilon - 1}$$

$$= n^{\log_b a - \varepsilon} \frac{n^\varepsilon b^\varepsilon - 1}{b^\varepsilon - 1} \leq n^{\log_b a - \varepsilon} \frac{n^\varepsilon b^\varepsilon}{b^\varepsilon - 1} = n^{\log_b a} \frac{b^\varepsilon}{b^\varepsilon - 1}$$

$$T(n) = \underline{O(n^{\log_b a})}.$$

Case (2)

$$\sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a} = n^{\log_b a} \sum_{i=0}^{\log_b n} a^i b^{-i \log_b a} = n^{\log_b a} \sum_{i=0}^{\log_b n} a^i a^{-i}$$

$$= n^{\log_b a} (\log_b n + 1) = \Theta(n^{\log_b a} \log n),$$

combining this with (ii) and the assumption of (2), to within constant factor bounds we have.

$$T(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i) + O(n^{\log_b a}) = \sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a} + O(n^{\log_b a})$$

$$= \Theta(n^{\log_b a} \log n) + O(n^{\log_b a}) = \Theta(n^{\log_b a} \log n).$$

Case (3)

The lower bound is immediate, because $f(n)$ is a term of the sum (ii). For the upper bound, we will use the smoothness condition. This condition is satisfied by $f(n) = n^{\log_b a + \varepsilon}$ for any $\varepsilon > 0$ with $c = b^{-\varepsilon} < 1$:

$$af(n/b) = a(n/b)^{\log_b a + \varepsilon} = an^{\log_b a + \varepsilon} b^{-\log_b a} b^{-\varepsilon} = \underline{f(n) b^{-\varepsilon}}$$

In this case, we have $a^i f(n/b^i) \leq c^i f(n)$ (easy induction on i using the smoothness condition), therefore

$$T(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i) + O(n^{\log_b a}) \leq \sum_{i=0}^{\log_b n} c^i f(n) + O(n^{\log_b a}).$$

$$\leq f(n) \sum_{i=0}^{\infty} c^i + O(n^{\log_b a}) = f(n) \frac{1}{1-c} + O(n^{\log_b a}) = \underline{O(f(n))}.$$

Hence Proved.

b)

Ans- We can use unordered_map (which is a Hashmap)
For Hashmaps, the time complexity of add() and contain() function is O(1).
So, we will first initialise a hashmap 'm'.
We will go through each element of set A first.

```
for i from 0 to n-1.
    m[A[i]] = 1          /* sets all values to 1 in hashmap
                            for those key values present in A */
```

This will take O(n) time complexity and O(n) space complexity.
We will check set B,

```
for i from 0 to n-1
    if m[B[i]] == 1
        print 'Not Disjoint'
        Exit
print 'Disjoint'.
```

Here, we will check the value in Hashmap for a key corresponding to an element in set B, is 1 or not.
If it is 1, this means it is also present in set A and if none of elements in set B give 1 as value for their key, then the sets are disjoint.

Time Complexity :	$T(n) = O(n)$
Space Complexity :	$S(n) = O(n)$