Esra Eryılmaz
171044046

- CSE 321 -

Homework 3

Deadline: 24.12.2020

1) **Solve the following recurrence relation and give Θ relation for each of them.**

a) T(n)=27 T(n/3) +n^2

b) T(n)=9 T(n/4) +n

c) T(n)=2 T(n/4) +$\sqrt{n}$

d) T(n)=2 T($\sqrt{n}$) +1

e) T(n)=2T(n-2), T(0)=1, T(1)=1

f) T(n)=4T(n/2)+n, T(1)=1

g) T(n)= 2 T($\sqrt[3]{n}$)+1 , T(3)=1;

MASTER THEOREM

If $T(n) = a \cdot T(\frac{n}{b}) + f(n)$ $T(1)=c$ where $a \geq 1$, $b \geq 2$, $c>0$

If $f(n) \in \Theta(n^d)$ where $d \geq 0$, then

$$T(n) = \begin{cases} \text{case 1} \ \Theta(n^d) & \text{, if } a < b^d \\ \text{case 2} \ \Theta(n^d \cdot \log n) & \text{, if } a = b^d \\ \text{case 3} \ \Theta(n^{\log_b a}) & \text{, if } a > b^d \end{cases}$$

by Master Theorem;

a) $T(n) = 27\,T(n/3) + n^2$

$a=27$, $b=3$, $d=2$ $\rightarrow$ $a > b^d \Rightarrow 27 > 3^2$ case 3 is valid for this problem.

So the result is $\Theta(n^{\log_b a}) = \Theta(n^{\log_3 27}) = \Theta(n^3)$

b) $T(n) = 9\,T(n/4) + n$

$a=9$, $b=4$, $d=1$ $\rightarrow$ $a > b^d \Rightarrow 9 > 4^1$ case 3 is valid.

So the result is $\Theta(n^{\log_b a}) = \Theta(n^{\log_4 9}) = \Theta(n^{\log_{2^2} 3^2})$

$= \Theta(n^{\log_2 3})$ //

Scanned with CamSc

c) $T(n) = 2T(n/4) + \sqrt{n}$

$a=2$, $b=4$, $d=1/2$ $\rightarrow$ $a=b^d \Rightarrow 2 = 4^{1/2}$ case 2 is valid.

So the result is $\Theta(n^d \log n) = \Theta(n^{1/2} \cdot \log n) = \Theta(\sqrt{n} \cdot \log n)$ //

---

d) $T(n) = 2T(\sqrt{n}) + 1$

→ Let $\boxed{n = 2^k}$ then, $\boxed{T(2^k) = 2 \cdot T(2^{k/2}) + 1}$

→ Now, let $T(2^k) = S(k)$, then $T(2^{k/2}) = S(k/2)$

→ So we have; $\boxed{S(k) = 2 \cdot S(\frac{k}{2}) + 1}$

→ Now we can use Master Theorem;

$a=2$, $b=2$, $d=0$ $\rightarrow$ $a > b^d \Rightarrow 2 > 2^0$ case 3 is valid

$\Theta(k^{\log_2 2}) = \boxed{\Theta(k)}$

→ Now we have $n = 2^k$

So $\boxed{\log_2 n = k}$ $\longrightarrow$ using that we can get

$\Theta(k) = \Theta(\log_2 n) = \Theta(\log n)$ //

---

e) $T(n) = 2T(n-2)$, $T(0) = 1$, $T(1) = 1$

$T(n) = 2T(n-2)$
$T(n-2) = 2 \cdot T(n-4)$ $\longrightarrow$ $T(n) = 2^2 \cdot T(n - 2 \cdot 2)$

$\boxed{T(n) = 2^k \cdot T(n-2k)}$

if we assume $T(0) = 1$ and $k = \frac{n}{2}$

$T(n) = 2^{n/2} \cdot T(0)$

$T(n) = 2^{n/2}$

$T(n) = \Theta(\sqrt{2}^n)$ //

f) $T(n) = 4T(n/2) + n$, $T(1) = 1$

$a = 4$
$b = 2$
$d = 1$
$\rightarrow a > b^d \Rightarrow 4 > 2^1$ case 3 is valid.

So the result is $\theta(n^{\log_b a}) = \theta(n^{\log_2 4}) = \theta(n^2)$

---

g) $T(n) = 2 \cdot T(\sqrt[3]{n}) + 1$, $T(3) = 1$

→ Let's assume $\boxed{n = 2^k}$ $\Rightarrow$ $\boxed{T(n) = 2 \cdot T(2^{k/3}) + 1}$

→ Now let assume $T(2^k) = S(k)$, then $T(2^{k/3}) = S(k/3)$

→ So we have $\boxed{S(k) = 2 \cdot S(k/3) + 1}$

→ Now, we can use master theorem;

$a = 2$
$b = 3$
$d = 0$
$\rightarrow a > b^d \Rightarrow 2 > 3^0$ case 3 is valid.

$\theta(k^{\log_b a}) = \boxed{\theta(k^{\log_3 2})}$

→ Now, we have $n = 2^k$

So $\boxed{\log_2 n = k}$ $\longrightarrow$ using that we can get

$\theta(k^{\log_3 2}) = \theta(\log_2 n^{\log_3 2})$

2) How many lines (as a function of n) does the following program print? Write a recurrence relation and solve it by backward substitution. You may assume that n is a power of 2.

```
function f(n)
    if n <= 1:
        print_line("**")
    else:
        for i=1 to n
            f(n/2)
        end for
```

$T(1) = 1$

$f(n/2) \rightarrow T(n/2)$

It solves the problem by recursively, the subproblem of size $n/2$ and combine the solutions in linear time.

$$\boxed{T(n) = T(n/2) + n}$$

$$T(n) = T(n/2) + n$$
$$= T(n/4) + \frac{n}{2} + n$$
$$= T(n/8) + \frac{n}{4} + \frac{n}{2} + n$$
$$\vdots$$
$$T(n) = T\left(\frac{n}{2^k}\right) + \left(\frac{n}{2^{k-1}} + \cdots + \frac{n}{2} + n\right)$$

Assume $\boxed{n = 2^k}$

$$T(n) = \cancel{T(1)}^{\text{constant}} + \left(1 + 2 + 2^2 + \cdots + 2^{k-1} + 2^k\right)$$
$$= \frac{1 - 2^{k+1}}{1 - 2}$$
$$= 2^{k+1} - 1$$

$$T(n) = 2n - 1 \in \underline{\Theta(n)}$$

3) Let T(n) denote the worst case number of comparisons (A[0]>A[1]) made by the following function for an input array of n numbers. Give a recurrence relation for T(n). Solve the recurrence relation.

```
Algorithm Function_f (A[0..n-1])
//Input: Array A of n numbers
//Output: A is sorted in increasing order
if n=2 and A[0]>A[1], then swap(A[0],A[1])
if n>2 then {
    Function_f (A[0..ceil(2n/3)]).
    Function_f (A[floor(n/3)..n])
    Function_f (A[0..ceil(2n/3)])
}
```

$\rightarrow T(1) = 1$

$\rightarrow T(2n/3)$

$\rightarrow T(n - \frac{n}{3}) = T(2n/3)$

$\rightarrow T(2n/3)$

For the worst case number of comparisons,
The recurrence relation is;

$$\boxed{T(n) = \underbrace{3 \cdot T(2n/3)}_{\text{if } n>2 \text{ 3 recursive calls.}} + \underbrace{1}_{\text{if } n=2}} \qquad T(1) = 1$$

using master theorem;

$a = 3$
$b = 3/2$
$d = 0$

$\rightarrow a > b^d \Rightarrow 3 > \left(\frac{3}{2}\right)^0$ case 3 is valid.

So the result is $\Theta(n^{\log_b a}) = \Theta(n^{\log_{\frac{3}{2}} 3}) \approx \Theta(n^{2.7})$

4) Implement the quick sort and insertion sort algorithms and count the number of swap operations to compare these two algorithms. Analyze the average-case complexity of the algorithms. Compare the operations count in your report file to decide which algorithm is better and support your analysis by using the theoretical average-case analysis of your algorithms.

```
InsertionSort (A)
    for j=2 to length(A)
        key = A[j]
        i   = j-1
        while i>0 and A[i]>key
            A[i+1] = A[i]
                 i = i-1
        end while
        A[j+1]=key
    end for
end
```

```
QuickSort (A, low, high)
    if(low < high)
        pivot = partition(A, low, high, position)
        QuickSort(A, low, pivot-1)
        QuickSort(A, pivot+1, high)
    end if
end
```

```
Partition (A, low, high, position)
    pivot = A[low]
    right = low
    left  = high+1
    while (right < left)
        repeat right = right+1 until A[right] >= pivot
        repeat left = left-1 until A[left] <= pivot
        if (right < left)
            swap (A[left], A[right])
        end if
    end while
    position = left
    A[low] = A[position]
    A[position] = pivot
end
```

Average-Case Complexity of Insertion Sort:

Let T_i = Number of basic operations at step i.

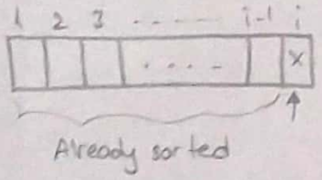
$$1 \le i \le n-1$$

$$A(n) = E[T] = E\left[\sum_{i=1}^{n-1} T_i\right]$$

- We need to calculate $E[T_i]$;

$$E[T_i] = \sum_{j=1}^{i} j \cdot \underbrace{P(T_i = j)}$$

→ Probability that there are j comparisons in the i^{th} step.



- 1 comparison will occur if $x = L[i] > L[i-1]$
- 2 comparison will occur if $L[i-2] < x < L[i-1]$
- 3 comparison will occur if $L[i-3] < x < L[i-2]$

⋮

- i comparison will occur if $L[1] < x < L[2]$
- i comparison will occur if $x < L[1]$

- There are $(i+1)$ intervals that x can fall in.
- There are $(i+1)$ case.

→ So $P(T_i = j) = \begin{cases} \dfrac{1}{i+1} & \text{if } 1 \le j \le i-1 \\ \dfrac{2}{i+1} & \text{if } j = i \end{cases}$

$$\rightarrow E[T_i] = \left[\sum_{j=1}^{i-1}\left(j \cdot \frac{1}{i+1}\right)\right] + \left(i \cdot \frac{2}{i+1}\right) = \frac{i(i+3)}{2\cdot(i+1)} = \frac{i}{2} + 1 - \frac{1}{i+1}$$

$$\rightarrow A(n) = E(T) = \sum_{i=1}^{n-1} E[T_i] = \sum_{i=1}^{n-1}\left(\frac{i}{2} + 1 - \frac{1}{i+1}\right) = \frac{n\cdot(n-1)}{4} + n - \underbrace{H(n)}_{\text{Harmonic series}}$$

$$A(n) \in \underline{\underline{\Theta(n^2)}}$$

Average-case complexity of Quick Sort:

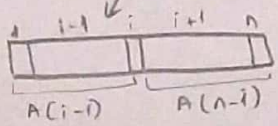
$$T = \underbrace{T_1}_{\text{Number of operations in rearrange.}} + \underbrace{T_2}_{\text{Number of operations in recursive call}}$$

$$A(n) = E[T] = \underbrace{E[T_1]}_{\text{high-low+2 operations}} + \underbrace{E[T_2]}_{\text{depends on where the pivot has been placed.}}$$

→ $E[T_2] = \sum_{x} E[T_2 \mid \bar{X} = x] \cdot \underbrace{P(\bar{X} = x)}_{\frac{1}{n}}$

position of the pivot (x)

→ $A(n) = \underbrace{(n+1)}_{\substack{E[T_1] \\ (\text{high-low}+2)}} + \sum_{i=1}^{n} \underbrace{E[T_2 \mid \bar{X} = i]}_{} \cdot P(\bar{X} = i)$



$$= (n+1) + \underbrace{\sum_{i=1}^{n} [A(i-1) + A(n-i)]}_{} \cdot \frac{1}{n}$$

$A(0) + A(n-1)$
$+ A(1) + A(n-2)$
$+ A(2) + A(n-3)$

$+ A(n-1) + A(0)$

$2[A(0) + \cdots + A(n-1)]$

$$A(n) = (n+1) + \frac{2}{n} \cdot [A(0) + \cdots + A(n-1)]$$

$$n \cdot A(n) = n \cdot (n+1) + 2 \cdot [A(0) + A(1) + \cdots + A(n-1)]$$
$$-\ (n-1) \cdot A(n-1) = n \cdot (n-1) + 2 \cdot [A(0) + A(1) + \cdots + A(n-2)]$$

$$n \cdot A(n) - (n-1) \cdot A(n-1) = 2n + 2A(n-1)$$

↓

$$\frac{1}{n\cdot(n+1)} \Big/ \quad \frac{A(n)}{n+1} - \frac{A(n-1)}{n} = \frac{2}{n+1} \Rightarrow \frac{A(n)}{n+1} = \frac{A(n-1)}{n} + \frac{2}{n+1}$$

→ Change of variable : $t(n) = \frac{A(n)}{n+1}$ → $\underbrace{t(n) = t(n-1) + \frac{2}{n+1}}$

It is a first order recurrence relation

By backword substitution:

$$t(n) = \sum_{i=2}^{n} \frac{2}{i+1} = 2\cdot H(n+1) - 3$$

Harmonic series

$$A(n) = t(n)\cdot(n+1) = 2\cdot(n+1)\cdot \underbrace{H(n+1)}_{\ln(n+1)} - 3(n+1) \in \underline{\underline{\Theta(n\log n)}}$$

Compare algorithms:

Theoretically, QuickSort is better than the InsertionSort.

Because QuickSort → $\Theta(n\log n)$

InsertionSort → $\Theta(n^2)$

If we look at the number of swap operations, we can say that number of swap operations are similar with the theoretical analysis.

Example : Arr = [1, 5, 10, 4, 3, 8, 9, 2]

Insertion Sort → 13

Quick Sort → 6

Number of swap operations

5) **What are the running times of each of these algorithms (in big-O notation), and which would you choose?**

a) An algorithm that divides the problem into 5 subproblems where the size of each subproblem is one third of the original problem size, solves each subproblem recursively and then combines the solutions to the subproblems in quadratic time.

b) An algorithm that divides the problem into 2 subproblems where the size of each subproblem is half of the original problem size, solves each subproblem recursively and then combines the solutions to the subproblems in $O(n^2)$ time.

c) An algorithm that solves the problem by recursively solving the subproblem of size n-1 and then combine the solutions in linear time.

a) $\boxed{T(n) = 5 \cdot T(n/3) + O(n^2)}$

By using master Theorem:

$\left.\begin{array}{l} a=5 \\ b=3 \\ d=2 \end{array}\right\} \rightarrow a < b^d \Rightarrow 5 < 3^2$ case 1 is valid.

So $\Rightarrow O(n^2)$ //

b) $\boxed{T(n) = 2 \cdot T(n/2) + O(n^2)}$

master Theorem;

$\left.\begin{array}{l} a=2 \\ b=2 \\ d=2 \end{array}\right\} \rightarrow a < b^d \Rightarrow 5 < 2^2$ case 1 is valid.

So $\Rightarrow O(n^2)$ //

c) $\boxed{T(n) = T(n-1) + O(n)}$

$\left.\begin{array}{l} T(n) = T(n-1) + O(n) \\ T(n-1) = T(n-2) + O(n-1) \end{array}\right\} \rightarrow T(n) = T(n-2) + n + n - 1$

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

$\vdots$

Assume $k=n$, $T(0)=1$

$\underbrace{\cancel{T(0)}}_{\text{constant}} + n^2 - \frac{n^2+n}{2} \in O(n^2) \quad = T(n-k) + k \cdot n - \frac{k \cdot (k+1)}{2}$

So all algorithms have the same complexity.

Any one can be chosen. //