CSE 321 Homework #3

Muhammed Yasir Fidan
161044056

1-) Master theorem:

$X(n) = a.X\left(\frac{n}{b}\right) + f(n)$

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

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

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

$f(n) \in \Theta(n^2)$ so $d = 2$

$a = 27$, $b = 3$

$27 > 3^2$ so; case 3

$T(n) \in \Theta(n^{\log_b^a}) = \Theta(n^{\log_3 27})$

$\underline{T(n) \in \Theta(n^3)}$

b) $T(n) = 9\, T\left(\frac{n}{4}\right) + n$

$a = 9$, $b = 4$, $d = 1$

$9 > 4^1$ so; case 3

$T(n) \in \Theta(n^{\log_4^9})$ $\quad \log_4 9 \approx 1.58$

$\underline{T(n) \in \Theta(n^{1.58})}$

c) $T(n) = 2\, T\left(\frac{n}{4}\right) + \sqrt{n}$

$a = 2$, $b = 4$, $d = \frac{1}{2}$

$2 = 4^{\frac{1}{2}}$ so; case 2

$\underline{T(n) \in \Theta(\sqrt{n} . \log n)}$

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

lets say $n = 2^k$

$T(2^k) = 2\,T(2^{k/2}) + 1$ and $T(2^k) = F(k)$

$F(k) = 2\,F(k/2) + 1$

now apply this master theorem

$a = 2,\ b = 2,\ d = 0$

$2 > 2^0$ so; case 3

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

$F(k) \in \Theta(k)$ but we must find $T(n)$

if $n = 2^k$ $k = \log_2 n$ so; $F(k) \in \Theta(\log_2 n)$

and because $T(n) = F(k)$, $\underline{T(n) \in \Theta(\log_2 n)}$

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

$T(2) = 2\,T(0) = 2^1$

$T(3) = 2\,T(1) = 2^1$

$T(4) = 2\,T(2) = 2^2$

$T(5) = 2\,T(3) = 2^2$

$\vdots$

$T(n) = 2\,T(n-2) = 2^{\lfloor \frac{n}{2} \rfloor}$ so; $T(n) \in \Theta(\sqrt{2}^{\,n})$

f) $T(n) = 4T(\frac{n}{2}) + n, \quad T(1) = 1$

$a = 4, b = 2, d = 1$

$4 > 2^1$ so; case 3 $\quad T(n) \in \Theta(n^{\log_2 4}) = \Theta(n^2)$

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

lets say $n = 3^k$

$T(3^k) = 2T(3^{k/3}) + 1$ and $T(3^k) = F(k)$

$F(k) = 2T(k/3) + 1$

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

$2 > 3^0$ so case 3; $F(k) \in \Theta(k^{\log_3 2})$

Now convert $F(k)$ to $T(n)$

If $n = 3^k$, $k = \log_3 n$ so $F(k) \in \Theta((\log_3 n)^{\log_3 2})$

$\log_3 2 = 0{,}63$ and $F(k) = T(3^k) = T(n)$ so:

$T(n) \in \Theta((\log_3 n)^{0.63})$

2-) function f(n)

```
        if n<=1:
            print_line("++")

        else
            for i=1 to n          ———— n
                f(n/2)            ———— T(n/2)
            end for
```

So this algorithm $T(n)=n.T(\frac{n}{2})$, $T(1)=1$

lets solve this by using back

$$T(n)=n.T(\frac{n}{2})$$

$$=n.\frac{n}{2}.T(\frac{n}{4})$$

$$=\frac{n}{2^0}.\frac{n}{2^1}.\frac{n}{2^2}.T(\frac{n}{2^3})$$

$$=\frac{n^k}{2^{\frac{k(k-1)}{2}}}.T(\frac{n}{2^k}) \qquad \frac{n}{2^k}=1,\ n=2^k,\ k=\log_2 n$$

$$T(n)=\frac{n^k}{2^{\frac{k(k-1)}{2}}}.T(1)=\frac{n^{\log_2 n}}{2^{\frac{(\log_2 n-1)\log_2 n}{2}}}$$

Now we need prove this, firstly check for $n=1$

$$T(1)=\frac{1^{\log_2 1}}{2^{\frac{(\log_2 1-1)\log_2 1}{2}}}=\frac{1^0}{2^0}=1 \quad \checkmark \text{ True}$$

Assume $T(n)$ is true

$$T(n) = \frac{n^{\log_2 n}}{2^{\frac{(\log_2 n - 1)\log_2 n}{2}}}$$

Now prove $T(2n)$ is also true

$$T(2n) = 2n \cdot T(n) = 2n \cdot \frac{n^{\log_2 n}}{2^{\frac{(\log_2 n - 1)\log_2 n}{2}}} \quad (1)$$

$$\text{Also } T(2n) = \frac{(2n)^{\log_2 2n}}{2^{\frac{(\log_2 2n - 1)\log_2 2n}{2}}}$$

$$= \frac{2n \cdot n^{\log_2 2n}}{2^{\frac{\log_2 n \cdot (\log_2 n + 1)}{2}}} = \frac{2n^2 \cdot n^{\log_2 n}}{2^{\frac{\log_2 n (\log_2 n + 1)}{2}}} \cdot \frac{n^{-1}}{n^{-1}} \quad \text{and } n^{-1} = 2^{\log_2 n^{-1}}$$

$$= \frac{2n \cdot n^{\log_2 n}}{2^{\frac{\log_2 n (\log_2 n + 1)}{2}} \cdot 2^{\log_2 n^{-1}}} = \frac{2n \cdot n^{\log_2 n}}{2^{\frac{\log_2 n (\log_2 n - 1)}{2}}} \quad (2)$$

Because of (1) and (2) equal we prove this recurrence. so;

$$T(n) = \frac{n^{\log_2 n}}{2^{\left(\frac{(\log_2 n - 1)\log_2 n}{2}\right)}}$$

3) Algorithm function_F (A[0.. n-1])

if n=2 and A[0]>A[1], then swap (A[0], A[1])

if n>2 then {

function_F (A[0... ceil $(\frac{2n}{3})$])

Function_F (A[floor($\frac{n}{3}$) ..... n])

Function_F (A[0.. ceil $(\frac{2n}{3})$]

}

This algorith divede array $\frac{2}{3}$ for every recursion call. So

its recursive call 3 times for $\frac{2}{3}$ part of array and sort it

so, we can show its recurrence relation as follow;

$$T(n) = T\left(\left\lceil \frac{2n}{3} \right\rceil\right) + T\left(\left\lfloor \frac{2n}{3} \right\rfloor\right) + T\left(\left\lceil \frac{2n}{3} \right\rceil\right) + 1$$

compare

first recursive call call function for left $\frac{2}{3}$ part of array

Second recursive call call function for right $\frac{2}{3}$ part of array

third recursive call call function again for left $\frac{2}{3}$ part of array

So; we can say

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

3 recursive call for $\frac{2}{3}$ part of array

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

and, by using master theorem we can solve this recurrence;

$a=3$ $b=\frac{3}{2}$, $d=0$

$3 > \frac{3}{2}^0$, $a > b^d$ case 3; $O(n^{\log_b a}) = O(n^{\log_{\frac{3}{2}} 3})$

$\log_{\frac{3}{2}} 3 \approx 2.71$

So; this algorith $\underline{T(n) \in O(n^{2.71})}$

4-) procedure Insertion Sort(L[1:n])

```
for i=2 to n do
    current ← L[i]
    pos ← i-1
    while (pos ≥ 1) and (current < L[pos])
        L[pos+1] ← L[pos]        → swap operation
        pos ← pos-1
    end while
    L[pos+1] ← current
  end for
end
```

Insertion sort average

We saw average case of insertion sort in class;

Let T_i = # of basic operation at step i, $1 \leq i \leq n-1$

$$T = T_1 + T_2 + \dots + T_{n-1} = \sum_{i=1}^{n-1} T_i$$

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

Here T_i is a random variable, its average expected number of comparison.

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

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

and
$$P(T_i = j) = \begin{cases} \frac{1}{i+1} & \text{if } 1 \leq j \leq i-1 \\ \frac{2}{i+1} & \text{if } j = i \end{cases}$$

$$E[T_i] = \sum_{j=1}^{i-1} j \cdot \frac{1}{i+1} + i \cdot \frac{2}{i+1}$$

$$= \frac{i(i-1)}{2(i+1)} + \frac{2i}{i+1} = \frac{i^2 - i + 4i}{2(i+1)} = \frac{i(i+3)}{2(i+1)}$$

$$= \frac{i}{2} + 1 - \frac{1}{i+1}$$

$$A(n) = E[T] = \sum_{i=1}^{n-1} E[T_i] = \sum_{i=1}^{n-1} \frac{i}{2} + 1 - \frac{1}{i+1}$$

$$= \underbrace{\frac{n(n-1)}{4}}_{\substack{\text{This part} \\ \text{complexity} \\ \Theta(n^2)}} + n - 1 - \underbrace{\sum_{i=1}^{n-1} \frac{1}{i+1}}_{\substack{\text{This is a harmonic serie so its} \\ \text{complexity } \Theta(\log n)}}$$

So, theorically insertion sort average case complexity $\Theta(n^2)$

```
procedure QuickSort(L[low:high])
    if high > low then
        call rearrange(L[low:high], position)
        call Quicksort(L[low: position-1])
        call Quicksort (L[position+1: high])
    endif

procedure rearrange(L[low:high], position)
    right = low
    left = high+1
    x = L[low]
    while (right < left)
        repeat right = right+1 until L[right] >= x
        repeat left = left-1 until L[left] <= x
        if (right < left)
            call swap (L[left], L[right])
        end if
    end while
```

Also, we learn Quick sort average case in class

Lets say;

$$\underbrace{T = T_1}_{\text{\# of Operation In partisipation}} + \underbrace{T_2}_{\text{\# of operation in recursive calls}}$$

$$A(n) = E[T] = E[T_1] + E[T_2]$$

this is always
high-low + 2
= n+1 operation

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

$$A(n) = (n+1) + \sum_{i=1}^{n} E[T_2 \mid X = i] \cdot \underbrace{P(X = i)}_{\frac{1}{n}}$$

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

$$A(n) = (n+1) + \frac{2}{n} \left[A(0) + A(1) + \dots A(n-1)\right]$$

$$n \cdot A(n) = n \cdot (n+1) + 2\left[A(0) + A(1) + \dots + A(n-1)\right]$$

$$(n-1) \cdot A(n-1) = n \cdot (n-1) + 2\left[A(0) + A(1) + \dots + A(n-2)\right]$$

$-$

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

$$= \frac{A(n)}{n+1} - \frac{A(n-1)}{n} = \frac{2}{n+1} \Rightarrow \frac{A(n)}{n+1} = \frac{A(n-1)}{n} + \frac{2}{n+1}$$

if we say $t(n) = \frac{A(n)}{n+1}$

$$t(n) = t(n-1) + \frac{2}{n+1}$$

$$t(0) = 0$$

by using backword substitution

$$t(n) = t(n-1) + \frac{2}{n+1}$$

$$= t(n-2) + \frac{2}{n} + \frac{2}{n+1}$$

$$= t(n-3) + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$

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

$$A(n) = (n+1)\, t(n) = 2(n+1) . H(n+1) - 3(n+1) \in \Theta(n \log n)$$

So; Theorically insertion sort average case $\Theta(n^2)$ and Quick sort average case $\Theta(n \log n)$. Theorically Quick sort average case seems better than insertion sort average case. Also in my implementation I create 50 random array that has length 10 and sort them with quicksort and Insertion sort. I find, for quick sort average 18.8 swap operation and 23.3 for insertion sort. So, as we seen in theorical analyz quicksort realy has a better average case than insertion sort. Nevertheless insertion sort close to quick sort so its not a bad algorithm

5-) 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.

This algorithm recurrence relation;

$$T(n) = \underbrace{5}_{\text{subproblem}}\, T\Big(\underbrace{\tfrac{n}{3}}_{\text{one third}}\Big) + \underbrace{n^2}_{\text{quadratic time}}$$

and we can solve this recurrence relation by using master theorem

$a=5,\ b=3,\ d=2$

$5 < 3^2$, so because of $a < b^d$ from master theorem

$T(n) \in \Theta(n^d)$ it means $T(n) \in \Theta(n^2) \in O(n^2)$

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.

This algorithm recurrence relation;

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n^2$$

by using master theorem we can solve this recurrence easily

$a=2,\ b=2,\ d=2,\quad 2 < 2^2$ so $a < b^d$ case from theorem

so $T(n) \in \Theta(n^d) \in (n^2)$ so $T(n) \in O(n^2)$

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

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

Lets use backward method to solve this

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

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

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

$$= T(n-4) + n + n + n + n - 1 - 2 - 3$$

$$\vdots$$

$$= T(n-k) + kn - \left(\frac{k.(k-1)}{2}\right)$$

if $k = n$ then

$$T(n-k) + kn - (k-1) = T(0) + n^2 - \left(\frac{n^2 - n}{2}\right)$$

$$= T(0) + \frac{2n^2 - n^2 + n}{2}$$

$$= T(0) + \frac{n^2 + n}{2}$$

its a constant

So; $T(n) = \frac{n^2+n}{2}$, now lets prove this by induction

$T(0) = 0$ ✓

Assume $T(n-1)$ is true

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

Now prove $T(n)$ is also true

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

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

$$T(n) = \frac{n^2 + n}{2}$$ so, it is proven $T(n)$ is true

Hence, $T(n) \in \Theta(n^2)$ also mean $\underline{T(n) \in O(n^2)}$

So; both a, b, c has $O(n^2)$ time complexity
we can choose any of them.