CSE 321 - Introduction to Algorithm Design

Homework 2

1801042103
Ozan GEÇKİN

Q1)

Rule) The Master Theorem applies to recurrences of the following form:

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function.

There are 3 cases:

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

2) If $f(n) = \Theta(n^{\log_b a} \log^k n)$ with $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

3) If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ with $\varepsilon > 0$, and $f(n)$ satisfies the regularity condition, then $T(n) = \Theta(f(n))$.

Regular condition: $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.

---

a) $T(n) = 16T(\frac{n}{4}) + n!$

$a = 16 \quad b = 4 \quad f(n) = n!$

Case 3

$\Omega(n^{\log_4 16 + \varepsilon}) \rightarrow n^{\log_4 16} = n^2$

$n! > n^2$

$T(n) = \Theta((f(n)) = \Theta(n!)$

---

b) $T(n) = \sqrt{2}\, T(\frac{n}{4}) + \log n$

$a = 2^{1/2}$

$b = 4$

$f(n) = \log n$

$n^{\log_b a} = n^{\log_4 \sqrt{2}} = n^{0.25}$

$\log n \quad n^{0.25}$

Case 1

$T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_4 \sqrt{2}}) = \Theta(n^{0.25})$

---

c) $T(n) = 8T(\frac{n}{2}) + 4n^3$

$a = 8 \quad b = 2 \quad f(n) = 4n^3$

$n^{\log_b a} = n^{\log_2 8} = n^3$

Case 2

$T(n) = \Theta(n^{\log_b a} \log n)$

$T(n) = \Theta(n^3 . \log n)$

d) $T(n) = 64T\left(\frac{n}{8}\right) - n^2 \log n$

180104210 3
Ozan GEÇKİN

Theorem doesn't apply because f(n) that is negative

e) $T(n) = 3T\left(\frac{n}{3}\right) + \sqrt{n}$

$a=3$ $f(n)=\sqrt{n}$ $\log_a^b = \log_3^3 = 1$ Case 1

$b=3$

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

$c = 1/2$

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

$a = 2^n \to$ not constant

$b=2$ Theorem does not apply

$f(n) = -n^n \longrightarrow$ f(n) is not asymptotically positive function

g) $T(n) = 3T\left(\frac{n}{3}\right) + \frac{n}{\log n}$

$a=3$

$b=3$ $a = b^d \Rightarrow 3 = 3^1$

$f(n) = n^d (\log n)^{-1}$ So, Case 2

$d=1$

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

Q2) What are the running times of each of these algorithms (in big-O notation), and which would you choose?

Rules

$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$ $a \rightarrow$ number of subproblems

$\frac{n}{b} \rightarrow$ size of the subproblems

a) Algorithm x

$T(n) = 9T\left(\frac{n}{3}\right) + n^2 \rightarrow$ quadratic time

Let's using master theorem:

$a = 9$ $f(n) = n^2$
$b = 3$

$\Rightarrow n^{\log_b^a} = n^{\log_3 9} = n^2$

$f(n) = \Theta(n^{\log_b^a})$

case 2

$T_n = \Theta(n^2 \log n)$

b) Algorithm y

$T(n) = 8T\left(\frac{n}{2}\right) + n^3$

Let's using master theorem:

$a = 8$
$b = 2$ $n^{\log_2 8} = n^3$
$f(n) = n^3$

case 2

$T(n) = \Theta(n^{\log_b a} \cdot \log n)$
$= \Theta(n^3 \cdot \log n)$

1801042103
Ozan GEÇKİN

1801042103
Ozan GEÇKIN

C) Algoritm z

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

Case 2

$a = 2$
$b = 4$
$f(n) = \sqrt{n}$

$n^{\log_b a} = n^{\log_4 2} = \sqrt{n}$

$T(n) = \Theta(n^{\log_b a} . \log n)$
$= \Theta(\sqrt{n} . \log n)$

Running times of algorithm

compare algorithm x at algorithm y

$\lim_{n \to \infty} \frac{n^2 \log n}{n^3 . \log n} = \frac{1}{n} = 0 \longrightarrow$ algorithm x faster than y

compare algorithm x at algorithm z

$\lim_{n \to \infty} \frac{n^2 \log n}{\sqrt{n} . \log n} = \frac{n^2}{\sqrt{n}} = \infty \longrightarrow$ algorithm z faster than x

compare algorithm y and z =

$\lim_{n \to \infty} \frac{n^3 \log n}{\sqrt{n} . \log n} = \frac{n^3}{\sqrt{n}} = \infty \longrightarrow$ algorithm z faster than algo y

Result;

According to fast = algo z > algo x > algo y

I would prefer Algorithm z because it is the fasthest algorithm.

Q3)

1801042103
Ozan GECKIN

a)

Rule)

merge sort is a divide and conquer algorithm. It divides the input array into two halves, calls itself for two halves and then merges the two sorted halves.

Time complexity = Best case = $\Omega(n\log n)$

Average case = $\Theta(n\log n)$

Worst case = $O(n\log n)$

i) Merge Sort maximum comparision = All pair comparisons must be in the wrong order,

worst case

8, 4, 12, 0, 14, 10, 6, 2

8,4, 12,0 — 14, 10, 6, 2

8,4 — 12,0 — 14,10 — 6,2

4,8 — 0,12 — 10,14 — 2,6

0,4,8,12 — 2,6,10,14

0,2,4,6,8,10,12,14

Every step, index changes.

ii) Merge Sort minimum comparsion = All pair comparisions therefore pairs are always sorted.

best case

0, 2, 4, 6, 8, 10, 12, 14

0,2,4,6 — 8,10,12,14

0,2 — 4,6 — 8,10 — 12,14

0,2 — 4,6 — 8,10 — 12,14

0,2,4,6 — 8,10,12,14

0,2,4,6,8,10,12,14

Not happens index changes

Note = Best case is the same as worst case because comparison must occur at all costs.

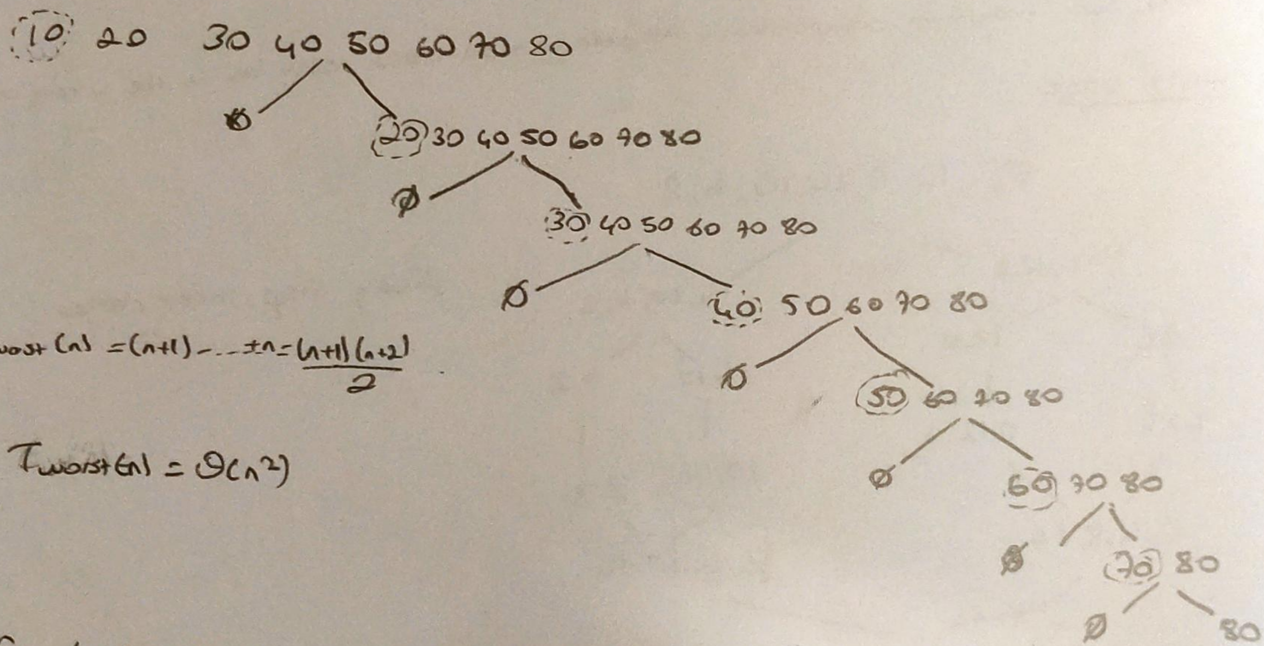
b) Rules) QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot.

Time complexity = Best case = $\Omega(n\log n)$

Average case = $\Theta(n\log n)$

Worst case = $O(n^2)$

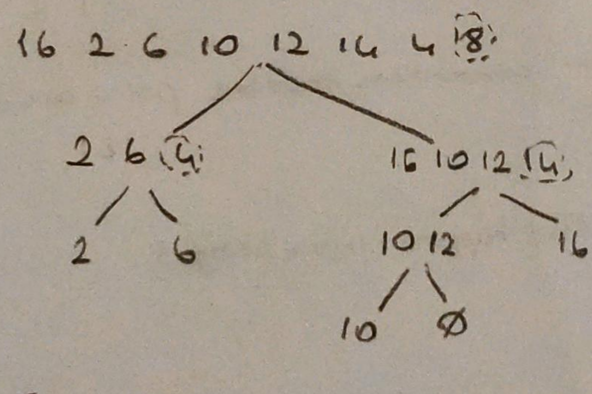
i) For worst case the list is already sorted in either ascending or descending order. So there is noting to compare to in one side



$T_{worst}(n) = (n+1) - \dots + n = \frac{(n+1)(n+2)}{2}$

$T_{worst}(n) = \Theta(n^2)$

ii) for best case split must happen in the middle



$T_{best}(n) = 2\,T_{best}\left(\frac{n}{2}\right) + n$, $T_{best}(n) = n\log n$

1801042103
Ozan GEÇİKİN

180104210 3
Oza GECKIN

Q4) algorithm(left, right)

```
algorithm(left, right)
    mid = (left + right)/2
    if A[mid] == 0
        return mid
    else
        if A[mid] > 0
            right = mid
            algorithm (left, right)
        else
            left = mid
            algorithm (left, right)
```

This algorithm is Binary Search algorithm. Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise, narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

This algorithm;
1. Check A[mid] == 0
2. Check A[mid] > 0, delete the upper half
3. Check A[mid] < 0, delete the lower half

Thus the recurrence relation → $T(n) = T\left(\frac{n}{2}\right) + 1$ → firstly step

$$T\left(\frac{n}{2^{k-1}}\right) = T\left(\frac{n}{2^k}\right) + k \rightarrow k^{th} \text{ step} \quad \text{last step } n = 1$$

$$n = 2^k \rightarrow \log n = k$$

$$T(n) = \underbrace{T\left(\frac{n}{2^{\log_2 n}}\right)}_{\text{base case}} + \log n$$

$$= T(1) + \log n$$

So, $T(n) = \Theta(\log n)$

Q5)

1801042103
Ozan GEÇKİN

a) Algorithm in pseudocode

```
matchs (gifts, boxs, low, high)
    if low < high
    pivot = portition (gifts, boxs, low, high, boxs[high]
    portition (boxs, low, high, gifts[pivot]
    Match (gifts, boxs, low, pivot-1)
    Match (gifts, boxs, pivot+1, high)

portition (arr, low, high, pivot)
    i = low
    J = low
  while( J < high
     if (arr[J] < pivot)
        arr[i], arr[J] = arr[J], arr[i]
        i += 1
     elif (arr[J] == pivot)
        arr[J], arr[high] = arr[high], arr[J]
        J -= 1
     J += 1
  arr[i], arr[high] = arr[high], arr[i]
  return[i]
```

b) This algorithm firstly a partition by selecting the last element of the boxes array as the pivot, rearranges the gift array, and rotates the potition index 'i' so that all gifts smaller than gifts[i] are on the left side and all gifts are large than gifts.[i] is on the right. We can then segment the boxes array using gifts[i]. This process also makes the string of gifts and boxs nicelly segmented when we divide both gifts and boxs, the total time complexity will be

Recurrence Relation

$T(0) = T(1)$

$T(N) = 2T\left(\frac{N}{2}\right) + N$

$\frac{T(N)}{N} = \frac{N}{N} + \frac{2T(N/2)}{N}$

$\vdots$

$\frac{T\left(\frac{N}{N-2}\right)}{\frac{N}{N-2}} = 1 + \frac{T\left(\frac{N}{N}\right)}{\frac{N}{N}} = 1 + T(1)$

$\frac{T(N)}{N} = 1 + 1 + 1 \cdots + 1 = \log N$

$\frac{T(N)}{N} = \log N$

$T(N) = N \log N$

$O(N \log N)$