Baran Solmaz
180104261

1.a) $T(n) = \underset{a}{16}\, T(\tfrac{n}{4}) + n!$ ($b = 4$)

$\frac{f(n)}{n!} \in O(n^d)$ $\quad d = \log_n n!$

$4^2 < 4^d$ ✓ $\quad$ d is greater than 2

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

$n^{\log_n n!} - n!$

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

$\log n$: k, p

$O(n^0 \cdot \log^1 n)$

$\log_4 \sqrt{2} = \frac{1}{4} > 0$ $\quad$ case 1

$T(n) = \Theta(n^{1/4})$

c) $T(n) = \underset{a}{8}\, T(n/2) + 4n^3$ ($b = 2$)

$k = 3$ $\quad p = 0$

$\log_2 8 = 3 \Rightarrow k$ $\quad$ case 2

$T_n = \Theta(n^3 \cdot \log n)$

d) $T(n) = 64\, T(n/8) - n^2 \log n$

not positive

so, Master Theorem doesnot apply

1.e) $T(n) = \underset{a}{3}\,T(n/\underset{b}{3}) + \sqrt{n} \Rightarrow (n^k \log^p n)$

$k = 1/2 \quad p = 0$

$\log_3 3 = 1 > 1/2$

$\boxed{T(n) = \Theta(n)}$

f) $T(n) = \underbrace{2^n}\, T(n/2) - n^n$

not constant.

Master theorem does not apply

g) $T(n) = \underset{a}{3}\,T(n/\underset{b}{3}) + \frac{n}{\log n} \quad (n^1 . \log^{-1} n)$

$\log_b a = 1 = k = 1 \quad p = -1 \Rightarrow \underline{\Theta(n . \log(\log n))}$

2) a) ~~X~~ → 9 sub problems, size $= n/3$, n^2,

Y → 8 subproblems, size $n/2$, n^3

Z → 2 subproblems, size $n/4$, $\sqrt{n}$

$X \Rightarrow T(n) = 9 . T(n/3) + n^2 \Rightarrow O(n^2 . \log n)$

$Y \Rightarrow T(n) = 8 . T(n/2) + n^3 \Rightarrow O(n^3 \log n)$

$Z \Rightarrow T(n) = 2 . T(n/4) + \sqrt{n} \Rightarrow O(\underset{\uparrow}{\sqrt{n}} . \log n)$

$\underline{Z < Y < X}$

Z is better algorithm because it is faster than others. less subproblems, less size and fast basic operation ($\sqrt{n}$)

3.a) Merge sort

i) Maximum

for maximum number of comparison, every sub problems must do maximum number of comparison.

first = a b c d e f g h

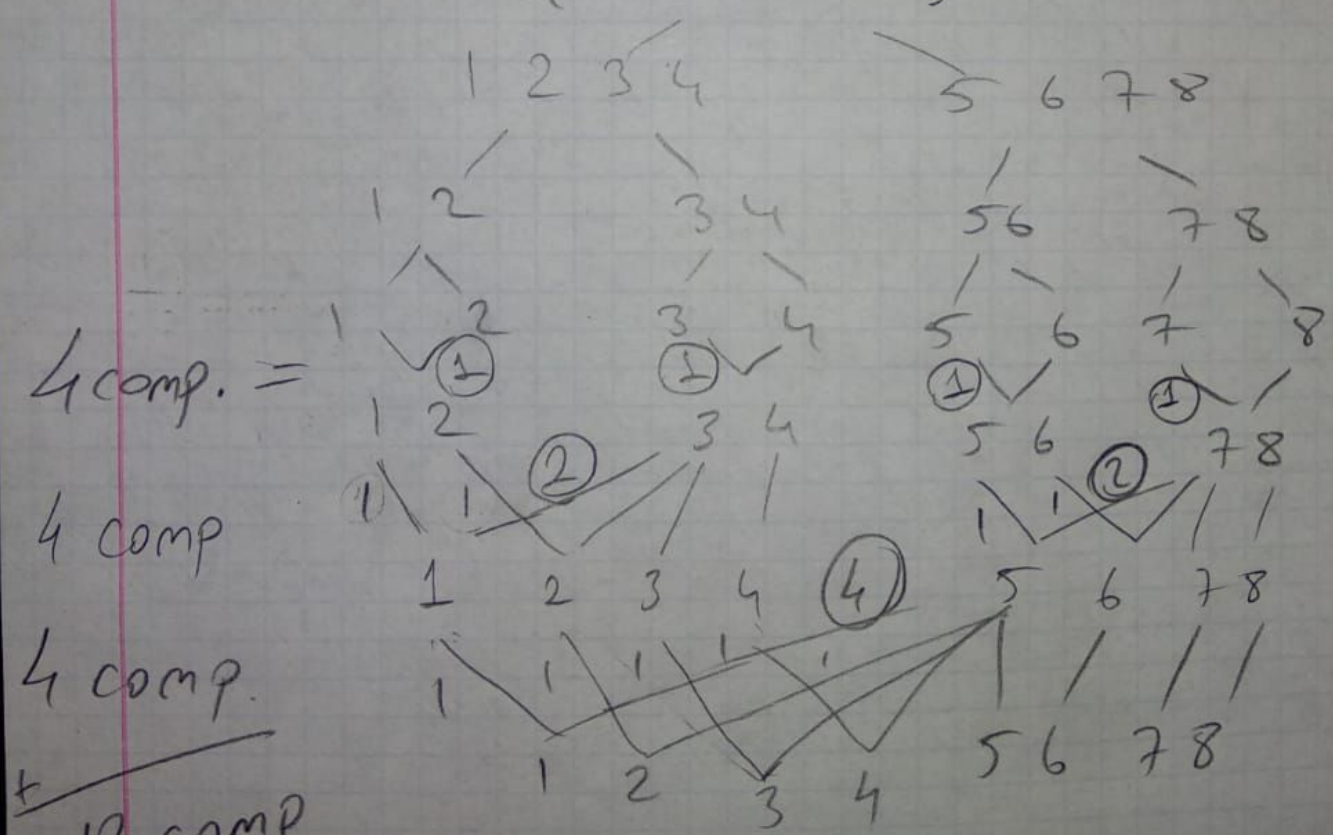
final = a e c g b f d h
1 2 3 4 5 6 7 8

Arr = {1,5,3,7,2,6,4,8}

** comparison = 17 comp

ii) Minimum

Arr = {1,2,3,4,5,6,7,8}



When the first array become empty, the seconds arrays elements are copied.

no comparison

3.5) Quick sort

i) maximum swap (first element pivot)

for maximum swap; every time when rearrenge function called, it must take the current element to the end of the array.

Arr = {1 - 8 - 2 - 7 - 3 - 6 - 4 - 5}

# of swaps : 25

ii) minimum swap (last element pivot)

for minimum swap; array must be already sorted.

Arr = {1, 2, 3, 4, 5, 6, 7, 8}

swaps : 0

4) algorithm(left, right)
mid = (left + right)/2 Θ(1)
if A[mid] == 0
return mid) Θ(1)
else
if A[mid] > 0
right = mid
algorithm(left, right)) T(n/2) + Θ(1)
else
left = mid
algorithm(left, right) } T(n/2) + Θ(1)

log n (T(n) = T(n/2) + Θ(2) → Only 1 time calls function due to if statement
T(n/2) = T(n/4) + Θ(2)
⋮
T(0) = Θ(2))

⇒ T(n) = log n . Θ(2)
= Θ(log₂ n)

5)
a)
prodPartition(A[low:high], pivot) ⇒ Θ(n)
i = low-1,
J = low.
while J <= high-1 do Θ(high-1-J)
if arr[J] <= pivot do
i++
swap(arr, i, J) Θ(1)
end if
J++
end while
swap(arr, i+1, high) Θ(1)
end

prodSort(Gifts[0-n], Box[0,n], 1)
if (i >= n)
return
end if
partition(Box, 0, n-1, gift(i))
Sort(gifts, box, i+1)
end

b) i=0 at the beginning
T(n) = T(n-1) + Θ(n)
T(1) = 1 ⇒ n . Θ(n) = Θ(n²)