CSE-321 Introduction to Alghorithm Design

Homework-2

Master Theorem

If $T(n) = aT(n/b) + f(n)$, $a >= 1$, $b > 1$

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & f(n) = O(n^{\log_b a - \varepsilon}) \\ \Theta(n^{\log_b a} \log n) & f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ and } af(n/b) < cf(n) \end{cases}$$

$\varepsilon > 0$ $c < 1$ $f(n)$ is increasing function

A-1)

a) $T(n) = 16T(n/4) + n!$

$a = 16$ $b = 4$

$f(n) = n! = \Omega(n^{\log_4 16 + \varepsilon})$, $16 \cdot (n/4)! < c \cdot n!$ True for large n

3rd case: $\Theta(f(n)) = \Theta(n!)$

($n!$ grows faster than n^2)

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

$a = \sqrt{2}$ $b = 4$

$f(n) = \log n = O(n^{\log_4 \sqrt{2} - \varepsilon})$ 1st case: $\Theta(n^{\log_4 \sqrt{2}}) = \Theta(\sqrt[4]{n})$

($\sqrt[4]{n}$ grows faster than $\log n$)

c) $T(n) = 8T(n/2) + 4n^3$

$a = 8$, $b = 2$, $f(n) = 4n^3 = \Theta(n^{\log_2 8})$.

2nd case: $\Theta(n^{\log_2 8} . \log n) = \Theta(n^3 \log n)$

(same growth rate)

d) $T(n) = 64T(n/8) - n^2 \log n$ → Does not apply because $F(n) = -n^2 \log n$ is non increasing.

e) $T(n) = 3T(n/3) + \sqrt{n}$ $a = 3, b = 3$

$f(n) = \sqrt{n} = O(n^{\log_3 3 - \varepsilon})$ 1st case $\Theta(n)$

f) $T(n) = 2^n T(\frac{n}{2}) - n^n$ → Does not apply beacuse:
- a is not constant
- $f(n) = -n^n$ is non-increasing.

g) $T(n) = 3T(n/3) + \frac{n}{\log n}$ → Does not apply because n is not polynomially larger than $n/\log n$

A-2) (solved using Master Theorem)

Alghoritm X: $9T(n/3) + n^2$ $f(n) = n^2 = \Theta(n^2)$ 2nd case: $\Theta(n^2 \log n)$ also, $O(n^2 \log n)$

Alghoritm Y: $8T(n/2) + n^3$ $f(n) = n^3 = \Theta(n^3)$ 2nd case: $\Theta(n^3 \log n)$ also, $O(n^3 \log n)$

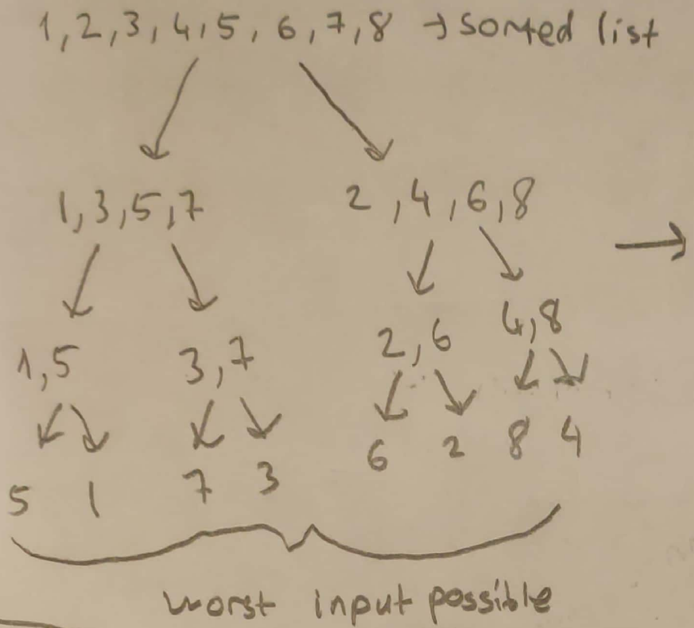
Alghoritm Z: $2T(n/4) + \sqrt{n}$ $f(n) = n^{1/2} = \Theta(n^{1/2})$ 2nd case: $\Theta(\sqrt{n} \log n)$ also, $O(\sqrt{n} \log n)$

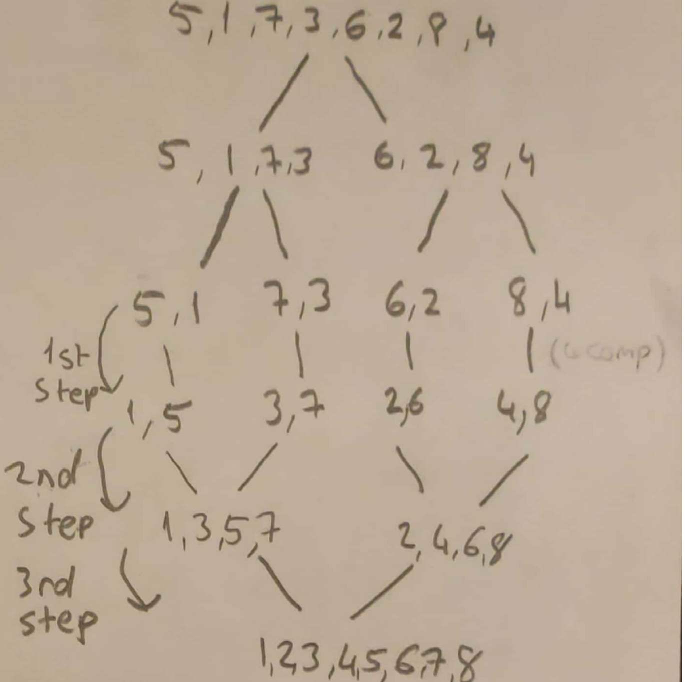
I would choose Alghorithm Z because it has the minimum complexity amongst all.

A-3)

a)

i) For maximum amount of comparisons, every pair of elements should be compared to each other at every step. To achieve this, the smallest should be on left, right, left, right... arrays at every check. Building the worst case with reversing merge sort:





Comparisons:

1st step: (5,1), (7,3), (6,2), (8,4) 4

2nd step: (1,3), (3,5), (5,7), (2,4), (4,6), (6,8) 6

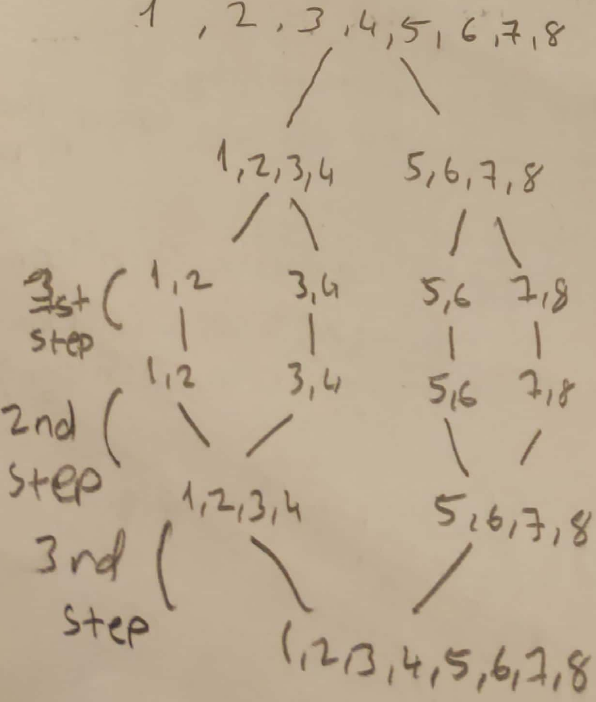
3rd step: (1,2), (2,3), (3,4), (4,5), (5,6), (6,7), (7,8) 7

+

max: 17

ii: for minimum amount of comparisons, list should be already sorted.

1,2,3,4,5,6,7,8



Comparisons:

1st step: (1,2), (3,4), (5,6), (7,8) 4

2nd step: (1,3) (2,3) (5,7), (6,7) 4

3rd step: (1,5) (2,5), (3,5) (4,5) 4

+

min: 12

6) I couldn't think of an algorithm to find the array with max number of swaps. I wrote a program which tries every comb. with 8 elements [0-7]. (Brute force)

And find 9 swaps at max [2,3,4,5,6,7,1,0]

A-4) Problem is divided by 2 and each subproblem has the half size of the main problem.

there are const. amount of operations for every recursion step.

$T(n)=2T(n/2)+1$ $f(n)=1=O(n^{1-\epsilon})$ 1st case: $\Theta(n)$

A-5)

```
func matchPairs(boxes, gifts, low, high):
    if(low<high):
        pivot = partition(boxes, low, high, gifts[high])
        Partitition(gifts, low, high, boxes[pivot])
        matchPairs(boxes, gifts, low, pivot-1)
        matchPairs(boxes, gifts, pivot+1, high)
```

pivot for gifts partition

pivot for partition (can be any element of the gifts)

complexity:

$T(n)=T(m)+T(n-m)+2\overset{\text{partition}}{O(n)}$

Depends on m.

If we assume it is always divided to 2 equal parts

$T(n)=2T(M/2)+2O(n)$

$f(n)=2n=\Theta(n)$ Case 2 of Master theorem: $O(n\log n)$