Seonhye Yang (sy3420) CSCI-GA 1170 Homework 2

Problem 1

· four-way Merge-Sort. Like 2-way mergesort, this one will be 4-way.

N=4, so we will have 4 sorted arrays.
sorting = n/4.

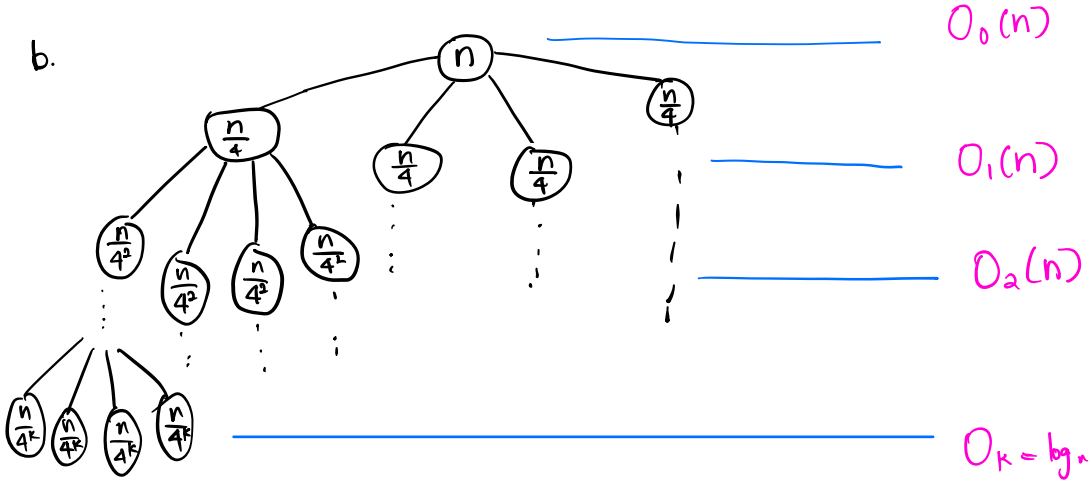
Two-way = $T(n) = 2T(n/2) + \Theta(n)$

$$T(n) = 4T(n/4) + 2n - 3$$
$$= 4T(n/4) + 2n$$
$$= (4^2 \cdot T(n/4^2) + 2n/4) + 2n$$
$$= 4^k \; T(n/4^k) + k(2n)$$

Let $\log_4(n) = k$

$$= N \cdot T(1) + (\log_4(n)) \cdot 2n$$
$$= N + 2N \cdot \log_4(n)$$
$$= n \log_2(n)$$

b.



$\Rightarrow O(n \cdot \log(n))$

$$T(n) = 4T(n/4) + 2n - 3$$ (throw away 3 because doesn't really matter)
$$= 4T(n/4) + 2n$$
$$= (4^2 \cdot T(n/4^2) + 2n/4) + 2n$$
$$= 4^k \; T(n/4^k) + k(2n)$$

Let $\log_4(n) = k$

$$= N \cdot T(1) + (\log_4(n)) \cdot 2n$$
$$= N + 2N \cdot \log_4(n)$$
$$= n \log_2(n)$$

Problem 2

a. 1. FIND(A, j, k, x)
2. If j > k Return False
3. Define i := (j + k)/2
4. If A[i] = i return True
5. If A[i] < i return FIND(A, j, i-1)
6. return FIND(A, i+1, k)

b. Time complexity:

$T(n) = T(n/2) + \Theta(1)$

$T(N/2) = T(n/2^2) + 1 + 1$

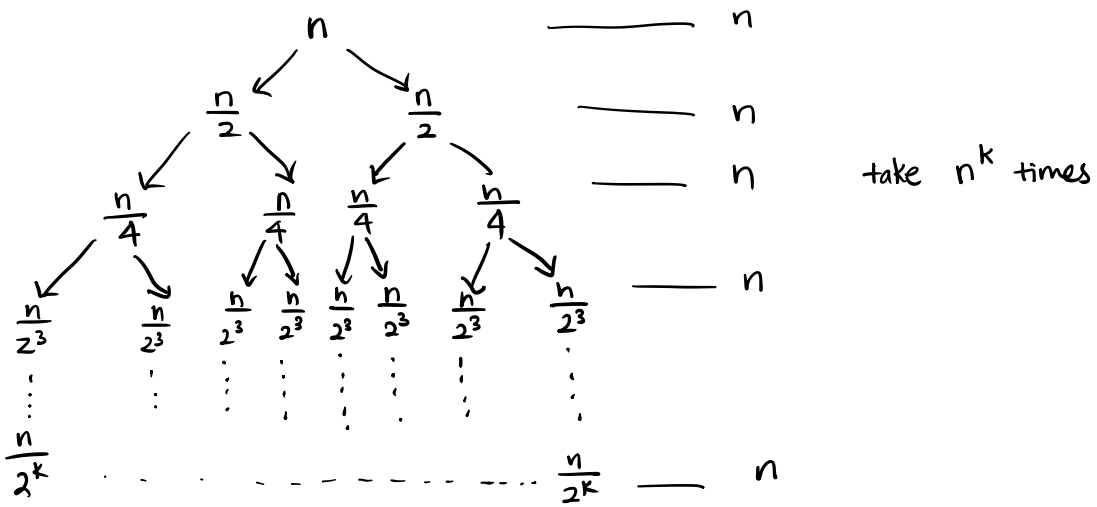
$T(N/k) = T(n/2^k) + 1 + \dots + 1$ (k)

Time complexity $= \Theta(\log(n))$

assume $\frac{n}{2^k} = 1 \Rightarrow n = 2^k$

$k = \log(n)$

Recursion Tree:



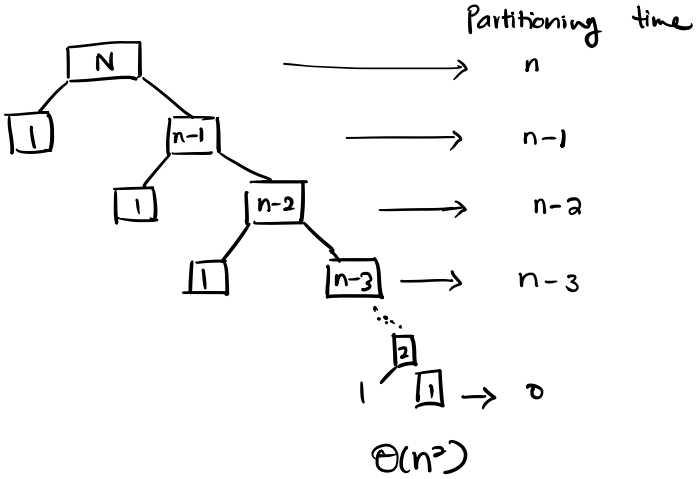
take n^k times.

- Assume $\frac{n}{2^k} = 1$

$n = 2^k$ and $k = \log(n)$

Problem 3

a. Given an array $A = [1, 2 \cdots n]$, we'll assume it's in some order ie 1,2,3,4...n. The running time is $O(n^2)$. This usually occurs when the array is sorted in some way. Also, the selection of the pivot matters. E.g. If we choose the leftmost element as a pivot, and the array is sorted, then we'd have two extremely unbalanced arrays.



$\Theta(n^2)$

b.
```
Partition ( A, first, mid, last, start):
    pivot = A[last]

    while mid[0] ≤ last :
        if (A[mid[0]] < pivot):
            A[mid[0]], A[start[0]] = A[start[0]], A[mid[0]]
            mid[0] = mid[0] +1
            start[0] = start[0] +1
        elif A(mid[0]] > pivot:
            A[mid[0]], A[last] = A[last], A[mid[0]]
            last = last -1
        else :
            mid[0] = mid[0] +1
```

c.
```
QuickSort (A, first, last):
    if first ≥ last :
        return
    if last == first +1 :
        if A[first > A[last] :
            A[first], A[last] = A[last], A[first]
            return
```

Time Complexity = $O(n \cdot \log(n))$.

- The average number of recursions made is $\log(n)$. And each time the function is called, it traverses the given array which takes $O(n)$ time. Therefore, the running time is $O(n \cdot \log(n))$.

Problem 4

a. An element occurs at least $n/4$ must be either $n/4^{th}$, $2n/4^{th}$, $3n/4^{th}$. This is because if there is an element that occurs at least $n/4$ times in array A, and if B is sorted array, A, and if i is the least index where x appears in B, then B[i] to B[i + n/4 - 1] are all equal to x.

e.g. A = [1, 1, 2, 3, 1, 4, 6, 7]

0 1 2 3 4 5 6 7

$\uparrow \frac{n}{4}$ $\uparrow \frac{2n}{4}$ $\uparrow \frac{3n}{4}$

$i = \frac{n}{4}^{th} = 2$

$\frac{2n}{4}^{th} = 4$

$\frac{3n}{4}^{th} = 6$

b.
```
Algos(A, n, k):
    dic = { }  # empty
    for i in range(n):
        if A[i] in dic:
            dic[A[i]] = dic[A[i]] + 1
        else:
            dic[A[i]] = 1
    x = n/k  # where k = 4

    for i in dic:
        if dic[i] > x:
            return i
```

where A = array, n = len(A), K = 4.