

# Assignment 3

Qiming Lyu

2025-10-13

1. (a) Each slice can be arranged in  $k!$  ways, and there are  $n/k$  such slices. Therefore, the number of permutations is

$$(k!)^{n/k}$$

- (b) The tree must have at least one leaf for each valid permutation, which gives

$$2^h \geq (k!)^{n/k}$$

Taking logarithm on both sides, we have

$$h \geq \frac{n}{k} \lg k!$$

- (c) We are given  $\lg n! = \Theta(n \lg n)$ , which implies

$$\lg k! = \Theta(k \lg k) = \Omega(k \lg k).$$

Therefore,

$$h = \Omega\left(\frac{n}{k} \lg k!\right) = \Omega\left(\frac{n}{k} \cdot k \lg k\right) = \Omega(n \lg k).$$

2. (a) Each pair of two elements could appear in two possible orders:

- either  $x < y$ ,
- or  $y > x$ .

We only allow  $x < y$  which is sorted correctly. Since there are  $n/2$  such pairs, we are cutting the number of possible arrangements by  $2^{n/2}$  in total. Thus, the number of valid permutations is

$$\frac{n!}{2^{n/2}}.$$

$$(b) 2^h \geq \frac{n!}{2^{n/2}} \implies h \geq \lg n! - \frac{n}{2}.$$

$$(c) h = \Omega\left(\lg n! - \frac{n}{2}\right) = \Omega(n \lg n - n) = \Omega(n \lg n).$$

3. The idea is to choose a good pivot at every step using a linear-time selection routine since the  $\Theta(n^2)$  time complexity only happens when the pivot is the smallest or largest element every time, which means we have unbalanced partitions every time. If we can always choose the median as the pivot, then we will always have two balanced partitions of size  $n/2$  each.

Say we are given a function "SELECT(A, k)":

```

1 function SELECT(A, k)
2   // A is the array to select from
3   // k tells to select the k-th smallest element
4   // Returns the k-th smallest element of A

```

Listing 1: Select

We are selecting the median of the array as the pivot. The partition routine is the same as that in quick sort. The overall algorithm is as follows:

```

1 function BETTER-QUICK-SORT(A)
2   // A is the array to be sorted
3   if A.length ≤ 1 then
4     return A
5   fi
6   p ← SELECT(A, A.length / 2)
7   (L, E, R) ← partition A around pivot p
8   return BETTER-QUICK-SORT(L) + E + BETTER-QUICK-SORT(R) // "+" is array
   concatenation
9 end

```

Listing 2: Better Quick Sort

To analyze the time complexity, we have the recurrence

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + \Theta(n) = 2T\left(\frac{n}{2}\right) + \Theta(n).$$

Therefore we have  $T(n) = \Theta(n \lg n)$ .

4. We can create an augmented array  $B$  by adding  $-\infty$  and  $+\infty$  so that the  $k$ -th smallest of the real items is the median of  $B$ .

Say we add  $x$  copies of  $-\infty$  and  $y$  copies of  $+\infty$ . Then the median index of  $B$  is

$$i_{\text{med}} = \frac{n + x + y + 1}{2} \quad (\text{one-indexed}).$$

We want  $k + x = i_{\text{med}}$ , which gives

$$y = 2k + x - 1 - n \quad \text{and} \quad x = y - 2k + 1 + n.$$

We choose  $x$  and  $y$  as follows:

- if  $k \leq (n + 1) / 2$ , set  $x = n + 1 - 2k$  and  $y = 0$ ;
- if  $k > (n + 1) / 2$ , set  $x = 0$  and  $y = 2k - 1 - n$ .

Pseudo code is as follows:

```

1 function SELECT-K-VIA-MEDIAN(A, k)
2   // A is the array to select from
3   // k tells to select the k-th smallest element
4   // Returns the k-th smallest element of A
5   n ← A.length
6   if k ≤ (n + 1) / 2 then
7     x ← n + 1 - 2 * k
8     y ← 0
9   else

```

```

10      x ← 0
11      y ← 2 * k - n - 1
12  fi
13
14  B ← A + array of x copies of -infinity + array of y copies of +infinity
15  return MEDIAN(B)
16 end

```

Listing 3: Find  $k$ -th Smallest

```

5↓ function findSet(x)
// Non-recursive version of findSet with path compression
6    root ← x
7
8    // Find the root
9    while parent[root] ≠ root do
10       root ← parent[root]
11    done
12
13    // Path compression
14    while x ≠ root do
15       next ← parent[x]
16       parent[x] ← root
17       x ← next
18    done
19
20    return root
21 end

```

Listing 4: Non-recursive findSet with Path Compression

6. WLOG, assume the size of the Disjoint Set is a power of two.

- (a) **makeSet phase:** we create  $n$  sets, each containing one element. The time complexity is  $\Theta(n)$ .
- (b) **union phase:** we always union equal ranks.

In each round, we halve the number of sets by unioning every pair of sets with the same rank by calling "union(a,b)", where  $a$  and  $b$  are roots of the pair of sets.

Since their ranks are equal, union-by-rank makes one the parent of the other and increases the parent's rank by 1. After  $\lg n$  rounds, there is only one set left with rank  $\lg n$ .

In each round, we do (# Disjoint Sets/2) unions, each costing  $\Theta(1)$ . Therefore, the time complexity is

$$\Theta(n) + \Theta\left(\frac{n}{2}\right) + \Theta\left(\frac{n}{4}\right) + \dots + \Theta(1) = \Theta(n).$$

- (c) **findSet phase:** we do findSet operations  $m$  times on a deepest leaf  $v$  in the final tree. Each operation costs  $\Theta(\lg n)$  since the height of the tree is  $\lg n$  and there is no path compression. Therefore, the time complexity is  $\Theta(m \lg n)$ .

The overall time complexity is

$$\Theta(n) + \Theta(n) + \Theta(m \lg n) = \Theta(n) + \Theta(m \lg n).$$

When  $m \gg n$ , the time complexity is  $\Theta(m \lg n)$ .