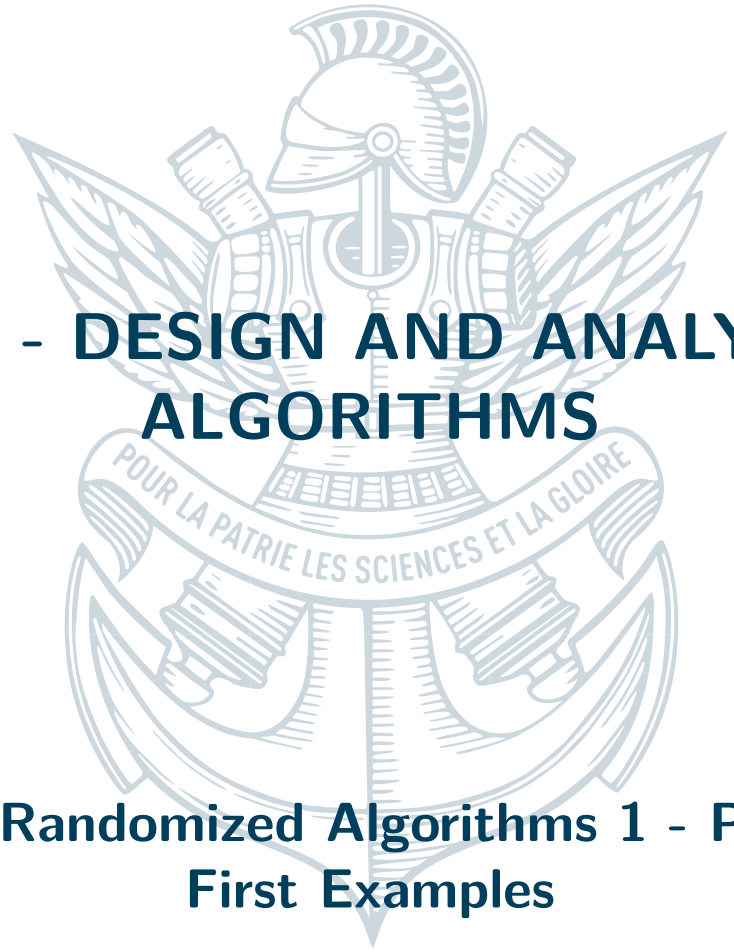

CSE202 - DESIGN AND ANALYSIS OF ALGORITHMS

Week 5 - Randomized Algorithms 1 - Principle & First Examples

October 23, 2022

Yubo Cai

EXERCISE 1

We have the partition program like the following:

```
def partition(A, lo, hi):
    p = A[lo]; i = lo; j = hi
    while True:
        for i in range(i + 1, hi):
            if A[i] > p:
                break
        for j in range(j - 1, lo - 1, -1):
            if A[j] <= p:
                break
        if i >= j:
            break
        A[i], A[j] = A[j], A[i]
    A[lo], A[j] = A[j], A[lo]
    return j
```

In the question, we have a constant number of distinct keys. First, we assume that there is 0 distinct keys which means all the keys are equal. if we change A[i] >= p to strictly bigger, then the index i would go to hi position directly, which requires comparing the whole loop. Since the index i runs directly to hi, which put pivot at the end of the subarray. In the worst case for quicksort, the recursive call will enter a subarray with length $n-1$ which would cause quadratic complexity.

Then we assume an array with length n and k distinct keys. This case is similar to the above one. Since we have k distinct keys, at least one key occurs at least $\frac{n}{k}$ times. Hence, when we do the recursive call of a subarray with length $\frac{n}{mk}$ (where m is a constant) we only have one kind of key. From our previous analysis, we know the complexity of the subarray is quadratic. We can have the complexity in $O(\frac{n^2}{(m_1k)^2} + ...\frac{n^2}{(m_ik)^2}) = O(n^2)$ comparisons since all of them is linear combination of quadratic complexity.

EXERCISE 2

Base on the algorithms of 3-way partition, the array will be divided into 3 parts

1. the keys that are smaller than the pivot, located in the first part of the array.
2. the keys that are equal to the pivot, located in the median part of the array.
3. the keys that are bigger than pivot, located in the end part of the array.

Since we know that there are only two distinct keys, the sort algorithm will done by only one partition since we got three parts with the middle part considered sorted and the other two-part

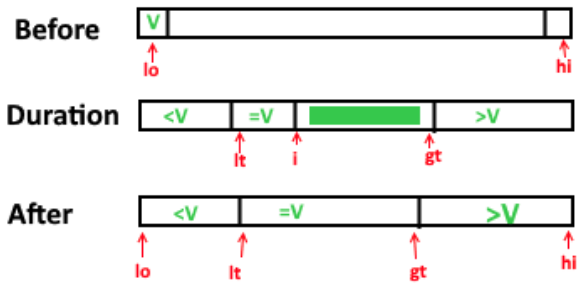

Figure 1: Division of the array in 3-way partition

one is empty and the other one only contains one kind of key. Therefore the array can be sorted by application of 3-way partition with complexity of $O(n)$.

EXERCISE 3

The situation above is compatible with the lower bound since in the question above we only have 2 distinct values in the array which cause the difference in the computation method. We obtained the bound by constructing balanced binary or ternary trees and calculating their nodes and levels which give us the complexity of $n \log n$. The previous question has specific restrictions about the different types of keys therefore we can get a complexity of $O(n)$ but not the general cases.