FTP_Alg_Week 4: Exercises and solutions

jungkyu.canci@hslu.ch

Exercise 1 *Prove that the worst case in Partitioning Algorithm (for Quicksort) has running time $\Theta(n^2)$, where n is the cardinality of the set of element of the partitioning.*

Solution: We start by giving an intuitive idea for calculating the running time. As we have seen in the lecture, we have the worst case, when at any step a k-size problem is divided with a $k-1$-elements array and an empty array. Let denote $T(n)$ the running time of Quicksort with the call of Partition. Since the splitting time of an array is linear in the number of element of the array we have

$$T(k) = T(k-1) + T(0) + \Theta(k) = T(k-1) + \Theta(k)$$

because $T(0)$ is constant and so $\Theta(1)$ and $\Theta(1) + \Theta(k) = \Theta(k)$ Now we iterative apply the above equality, obtaining:

$$\begin{aligned} T(n) &= T(n-1) + \Theta(n) \\ &= T(n-2) + \Theta(n-1) + \Theta(n) \\ &= T(n-3) + \Theta(n-2) + \Theta(n-1) + \Theta(n) \\ &\vdots \\ &= T(0) + \Theta(1) + \ldots + \Theta(n-1) + \Theta(n) \\ &= \Theta(1 + 2 + \ldots + (n-1) + n) = \Theta\left(\frac{n(n+1)}{2}\right) = \Theta(n^2) \end{aligned}$$

We have used the identity $1 + 2 + \ldots + (n-1) + n = \frac{n(n+1)}{2}$. This give an intuitive proof, which is not completely precise.

Now we give a proof by induction that the running time $T(n)$ in the worst case is $\Theta(n^2)$. We use the inductive hypothesis that for all $0 < m < n$, we have that $T(m) = \Theta(m^2)$. Recall that $T(m) = \Theta(m^2)$ means that there exists two positive integers such that c_1 and d_1 such that $c_1 m^2 \leq T(m) \leq d_1 m^2$. Furthermore we have that the partition has running time $P(m)$ that is $\Theta(m)$, that is there exist two constant c_2 and d_2 such that $c_2 m \leq P(m) + T(0) \leq d_2 m$. By considering $2c = \min\{c_1, c_2\}$ and $d = \max\{d_1, d_2, 1\}$, there fore we have

$$cm^2 < 2cm^2 \leq T(m) \leq dm^2 \quad \text{for all } m \geq n-1$$

and

$$2cm \leq T(0) + P(m) \leq dm \quad \text{for all } m \geq n$$

(Note that this last statement holds for all m, this is not an inductive hypothesis),

Now as above we have $T(n) = T(n-1) + T(0) + P(n)$ and since

$$c(n-1)^2 + 2cn = cn^2 - 2cn + 1 + 2cn = cn^2 + 1 > cn^2$$

and

$$d(n-1)^2 + dn = dn^2 - 2dn + 1 + dn = dn^2 - dn + 1 \leq dn^2$$

we have

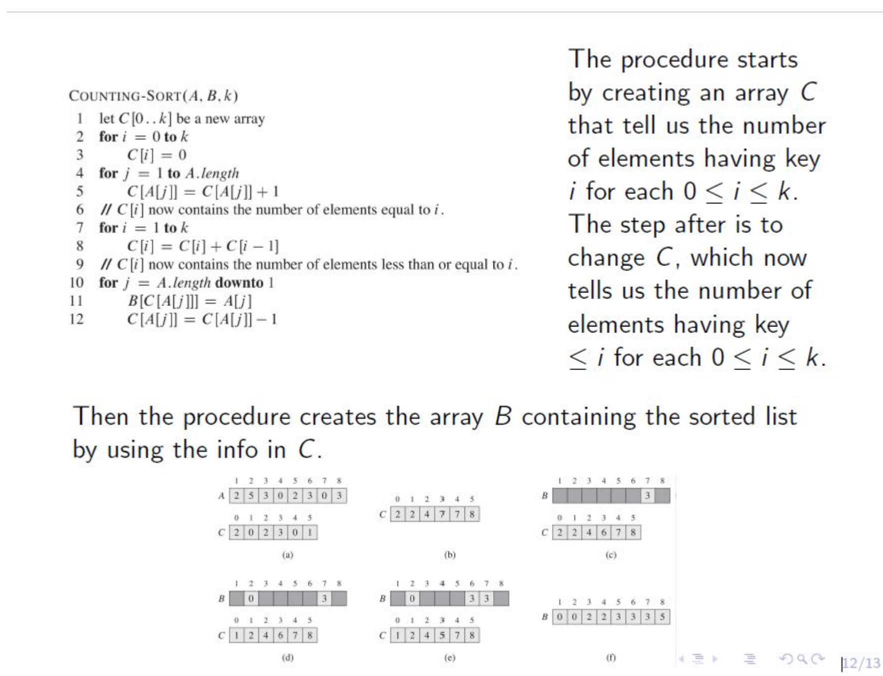
$$cn^2 < c(n-1)^2 + 2cn \leq T(n-1) + T(0) + P(n) = T(n) \leq d(n-1)^2 + dn \leq dn^2$$

that is what we had to prove.

Exercise 2 *We apply COUNTING–SORT (Lecture 7) with the input the vector*

$$A = (5, 6, 5, 3, 3, 7, 4, 4, 4, 5, 3, 8, 8)$$

Let C and B be the arrays mentioned in the pseudocode of COUNTING–SORT (below is the original algorithm from our lecture slides).



```
COUNTING-SORT(A, B, k)
1  let C[0..k] be a new array
2  for i = 0 to k
3      C[i] = 0
4  for j = 1 to A.length
5      C[A[j]] = C[A[j]] + 1
6  // C[i] now contains the number of elements equal to i.
7  for i = 1 to k
8      C[i] = C[i] + C[i - 1]
9  // C[i] now contains the number of elements less than or equal to i.
10 for j = A.length downto 1
11     B[C[A[j]]] = A[j]
12     C[A[j]] = C[A[j]] - 1
```

The procedure starts by creating an array C that tell us the number of elements having key i for each $0 \leq i \leq k$. The step after is to change C, which now tells us the number of elements having key $\leq i$ for each $0 \leq i \leq k$.

Then the procedure creates the array B containing the sorted list by using the info in C.

1. *Let C be the array obtained after the for loop at line 7 and 8 of the pseudocode (so the array at the step (b) of the figure at slide 12 of Lecture 7). The entry $C[7]$ is the number* 11

2. *We consider the first cycle **for** at line 10 of the pseudocode (see (c) of the figure at slide 12 of Lecture 7) in the figure. The number $B[13]$ is 8*

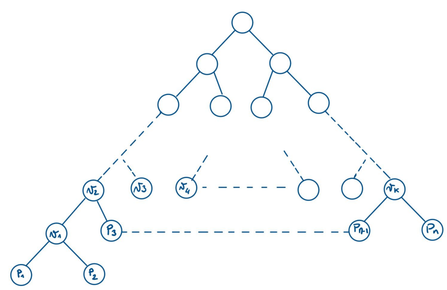
3. *We consider the first cycle **for** at line 10 of the pseudocode (see (c) of the figure at slide 12 of Lecture 7). At the end of the first cycle, the number $C[8]$ is 12*

Exercise 3 *Let n be a non negative integer and $P = \{p_1, p_2, \ldots, p_n\}$ n real numbers. Prove that there is a complete binary search tree having the points in P as leaves.*

Solution: We prove the statement by induction on the number n. The statement is trivially true for $n = 1$. Now suppose $n \geq 2$. Now by using Exercise 3 of Week 2, we have that with $N = 2n$ the leaves are in the node with indexes $\lfloor N/2 \rfloor + 1 = n + 1, \lfloor N/2 \rfloor + 2 = n + 2, \ldots, \lfloor N/2 \rfloor + n = 2n$. WLOG (without loss of generality) we can assume that the numbers $\{p_1, p_2, \ldots, p_n\}$ are sorted in increasing order

$$p_1 \leq p_2 \leq \ldots \leq p_n$$

(if not, reorder them and change the indexes). Now place the numbers in the leaves starting from p_1 in the far left as shown in the picture below



Now we should choose properly the keys $v_1, v_2, \ldots, v_k$ of the parents of the leaves (with key p_i), so that the Binary Search Tree Property is verified. We can take v_i any number verifying the property

$$lc(n_i).key \leq v_i \leq rc(n_i).key \tag{1}$$

where n_i is the node with $n_i.key = v_i$. It is clear that k (the number of parents of the leaves having keys the p_is) is less than n. Thus we can apply the inductive hypothesis and assuming that there exists a binary search tree having leaves with the keys $v_1, \ldots, v_k$. Therefore we can define the binary search tree defined as before, but before the nodes above the v_is were with no key. Condition (1) and inductive hypothesis assure that the binary search condition is verified.