1. Practice Problems (Chapter 2 of the text) 2.19, 2.21, 2.22, 2.23, 2.24, 2.32

Homework Problems

1. We are given an array $A$ with $n$ integer elements and a number $C$. Assume that the sum of the elements in $A$ is larger than $C$. We would like to compute the smallest subset of $A$ whose elements sum to at least $C$. (For example, if $A = [8,3,9,2,7,1,5]$ and $C = 18$, then the answer is $\{7, 8, 9\}$.) Give a linear expected time algorithm for this problem.

**Ans (hints):** After selecting the pivot and partitioning the array into two parts: $S_L$ and $S_R$, one can throw $S_L$ and work with $S_R$ if the total weight of $S_R$ is greater than $C$. When the total weight of $S_R$ is less than $C$, work with $S_L$ with new $C = \text{Old } C - \text{total weight of } S_R$. We then repeat the algorithm. The algorithm takes linear expected time.

2. We are given an array of integers $A[1..n]$. We would like to determine whether there exists an integer $x$ which occurs in $A$ more than $\frac{n}{3}$ times. Give an algorithm which runs in expected $O(n)$ time.

**Ans (hints):** The solution if exists is either the $\frac{n}{3}$ the smallest element or the $\frac{2n}{3}$ smallest element. Compute these two selected points (in linear expected time), and then check the number of times these two elements have appeared. The second step is linear.

3. We are given two arrays of integers $A[1..n]$ and $B[1..n]$, and a number $X$. Design an algorithm which decided whether there exist $i, j \in \{1, 2, \ldots, n\}$ such that $A[i] + B[j] = X$. Your algorithm should run in time $O(n \log n)$.

**Ans (hints):** We first sort the arrays. Set $a = 1$ and $b = n$. Check if $A[a] + B[b] = X$. Otherwise, if $A[a] + B[b] < X$ then $a = a + 1$, otherwise set $b = b - 1$.

4. Problems 2.17 and 2.24 from the text.

**Ans: Problem 2.17** This was discussed in the class. Assume that $n$ is even. After looking at the element $A[\frac{n}{2}]$ we can conclude the following:

- $(A[\frac{n}{2}] = \frac{n}{2})$: In this case we exit with the answer affirmative, i.e. yes for the index $\frac{n}{2}$.
• \((A[n^2] < \frac{n}{2})\): In this case the index \(i\) for which \(A[i] = i\) cannot exist in the sub-array \([1..n^2 - 1]\). If there exists such an \(i\), \(\frac{n}{2} - i < A[n^2] - i\) (why?), i.e. \(\frac{n}{2} < A[n^2]\), a contradiction.

• \((A[n^2] > \frac{n}{2})\): Using similar arguments we can show that an instance where \(A[i] = i\) cannot exist in the sub-array \([\frac{n}{2} + 1, .., n]\).

This way we are able to remove of of the total elements in every iteration.

Ans: Problem 2.24

(a) Modify the in place split procedure of 2.15 so that it explicitly returns the three subarrays \(S_L, S_R, S_v\). Quicksort can then be implemented as follows:

```
function quicksort(A[1..n])
    pick k at random among 1, .., n
    (S_L, S_R, S_v = split(A[1..n], A[k])
    quicksort(SL)
    quicksort(SR)
```

(b) In the worst case we always pick \(A[k]\) that is the largest element of \(A\). Then, we only decrease the problem size by 1 and the running time becomes \(T(n) = T(n - 1) + O(n)\), which implies \(T(n) = O(n^2)\).

(c) \(1 \leq i \leq n\), let \(p_i\) be the probability that \(A[k]\) is the \(i\)th largest element in \(A\) and let \(t_i\) be the expected running time cost of the algorithm in this case. The expected running time of the algorithm can be expressed as \(T(n) = \sum_{i=1}^{n} p_i t_i\). Since the probability of selecting any element of \(A\) as a pivot is the same, therefore \(p_i = \frac{1}{n}\). Moreover, \(t_i\) is at most \(O(n) + T(n - i + 1) + T(i - 1)\), as \(S_L\) has at most \(n - i + 1\) elements and \(S_R\) has at most \(i - 1\). Then

\[
T(n) \leq \frac{1}{n} \sum_{i=1}^{n} (T(n - i + 1) + T(i - 1) + O(n)) .
\]

i.e. \(T(n) \leq \frac{1}{n} \sum_{i=1}^{n-1} (T(n - i) + T(i) + O(n))\).

We can rewrite this for some constant \(c\):

\[
T(n) \leq cn + \frac{2}{n} \sum_{i=1}^{n-1} T(i)
\]

We can then show by induction that \(T(n) \in O(n \log n)\).