PROBLEM SET 6 – Manav Bilakhia

CSc 250, Spring 2014
Due: Thursday, Week 8

Aaron G. Cass
Department of Computer Science
Union College

Note: For each algorithm design question, you must prove that the algorithm works.

ALGORITHM DESIGN AND ANALYSIS

1. [From [1] pg. 177, #6.19] Given an array of integers $A[1..n]$, such that $A[1] < A[n]$ and $\forall i \in \{i \mid 1 \leq i < n\} : |A[i] - A[i+1]| \leq 1$.

   Design an efficient search algorithm to find j such that $A[j] = z$ for a given value z such that $A[1] \leq z \leq A[n]$. What is the maximal number of comparisons to z that your algorithm makes?

Answer:

WEIRDSEARCH($A[0...n], z$)
Input: takes in an array of size n and the element to be searched z
Output: returns the index of z

```
1: $low \leftarrow 0$
2: $high \leftarrow n-1$
3: while $low \neq up$ do
4:     low ← low +abs(val - A[low])
5:     high ← high -abs(val - A[high])
6:     mid ← (low+high)/2
7:     if A[mid] = z then
8:         return mid
9:     else if $z < A[mid]$ then
10:        high ← mid-1
11:    else
12:        low ← mid+1
13: return low
```

Let us assume without loss of generality that the given array has even number of elements. Therefore the variable mid takes the value A[n/2]. If $mid \geq z$ then z has to be in the array between A[0] and A[n/2]. This must hold true because the difference between consecutive elements of the array is at most 1. This condition makes sense as no number is skipped. For example, if the first element in the list is -5 and the last element in the list is 7 then we know that all integers between -5 and 7 must be there in the list. Numbers beyond this range could exist too. f numbers beyond this range exist then every number between the smallest element in the list and the largest element in the list must exist hence no number is skipped because of the constraints of the question i.e. the difference between two consecutive elements is at most 1. If $z > mid$ then by the same argument, z must be between the elements A[n/2 +1] and A[n]. This way in each case, our search space is cut by half with each comparison. Hence the maximal number of comparisons made is $log_2(n)$.

The running time of this algorithm would be $\Theta(logn)$ as this algorithm is similar to a simple binary search algorithm.

2. [From [1] pg. 178, #6.26] Given a list S of n real numbers, where n is even, a *pairing* P is a partition of S into $n/2$ pairs of elements, $\{(x_1, y_1), (x_2, y_2), ...(x_{n/2}, y_{n/2})\}$. Each pair (x_i, y_i) in a pairing has a sum, which is equal to $x_i + y_i$. The *maximal sum* of a pairing P is the largest sum of a pair in P. Therefore, a *minimal maximal sum pairing* of a set S is a pairing of S with the smallest maximal sum.

   So, given a list S of n elements, find a minimal maximal sum pairing P' of S, assuming n is even.

(a) Assume that S is sorted. Design a worst-case linear algorithm that solves the problem and prove that the worst-case running time is linear in n.

(b) Prove that the algorithm correctly solves the problem.

Answer:

(a) MMS($A[1...n]$)
Input: takes in an array of even length
Output: returns the pairing

```
1: if n=2 then
2:     return A[1],A[2]
3: for $i \leftarrow 0$ to $n/2$ do
4:     num1 ← A[i]
5:     num2 ←A[n-i-1]
6:     maxsum[i] ← num1, num2
7: return maxsum[]
```

(b) For this problem we know that n is always an even number and to be able to form a pair, $n \geq 2$. Let us first look at the case where $n = 2$. In this case there are only two elements in the array hence the minimal maximal sum pairing of this array would be the entire array. Let us now consider a case where $n > 2$. We know that there would be $n/2$ pairs in the pairing. To find the minimal maximum pair, we must pair the smallest number and the largest number in the array. Since the array is sorted we know that the first and the last element are the smallest and largest number respectively. If we pair the largest element with an element that is not the smallest then the sum would increase. Similarly if we pair the smallest number with another number which is not the largest then the pair that includes the largest element would not be optimal. Hence we pair the first element and the last element. After pairing the first and the last element, the largest unpaired element is the n-1 element and the smallest unpaired element is the 2nd element. Using the same argument as above, we pair these two. similarly we keep on pairing elements until we have exhausted the array.

The worst case running time of this algorithm is $\Theta(n)$

REFERENCES

[1] Udi Manber. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley, 1989.