Homework 3 Solution

1. *Given positive integers B, n and M as inputs, design an algorithm to compute the value of $(B^n \bmod M)$ in $O(\log n)$ time in the worst case.*

   Note that $B^{p+q} \bmod M = \left((B^q \bmod M) \times (B^q \bmod M)\right) \bmod M$.

   For any integer n can be rewrite into this form:

   $$n = a_0 \cdot 2^0 + a_1 \cdot 2^1 + a_2 \cdot 2^2 + a_3 \cdot 2^3 + \cdots + a_k \cdot 2^k$$

   Where

   $a_0, a_1, a_2, \cdots, a_k$ are either 0 or 1. And $k = \lfloor \log n \rfloor$.

   Algorithm:

   1) Compute $B^1 \bmod M$ and use the formula in the hints to calculate:
   2) $B^2 \bmod M, B^4 \bmod M, \cdots, B^{2^k} \bmod M$ where $k = \lfloor \log n \rfloor$

   Rewrite n as the form above and calculate regarding $a_0, \cdots, a_k$.

   3) Use the following formula to calculate $B^n \bmod M$

   $$B^n \bmod M = \left(a_0 \times (B^0 \bmod M)\right) \times \left(a_1 \times (B^1 \bmod M)\right) \times \cdots \times \left(a_k \times (B^k \bmod M)\right)$$

   Time complexity analysis:

   Step 1 takes $O(\log n)$. Step 2 takes $O(\log n)$. Step 3 takes $O(\log n)$. Thus the algorithm takes $O(\log n)$.

   (All the above Big O notation is on worst case)

2. *Suppose an array A consists of n elements, each of which is red, white, or blue. We seek to rearrange the elements so that all the reds come before all the whites, which come before all the blues. Your algorithm should run in $O(n)$ time in the worst case using $O(1)$ space.*

   In order to achieve the requirement, we use a double linked list to store the array.

   1) Initialize a pointer and point it to the start of the array
   2) Check the pointed element. If it is red, then move the pointer to the next element and move the checked element to the start of the array. Else, just move the pointer to the next element. Repeat this step until we reach the end of the array.
   3) Check the pointed element. If it is blue, then move the pointer to the previous element and move the checked element to the end of the array. Else, just move the pointer to the previous element. Repeat this step until we reach the start of the array.

   Time complexity analysis.

   Step 1 takes $O(1)$. Step 2 and 3 both takes $O(n)$. Thus the time complexity is $O(n)$.

   Space complexity analysis.

   Extra space we used is a pointer. So space complexity is $O(n)$.

3. *Given a list of n element A, outline an $\mathrm{O(n)}$ algorithm by modifying **Partition** algorithm discussed in class such that upon completion of the algorithm, there exists two index variables i and $\mathrm{j}(1 \le \mathrm{i} < \mathrm{j} \le \mathrm{n})$ such that $\mathrm{A(i+1)} = \mathrm{A(i+2)} = \cdots = \mathrm{A(j)};\ \mathrm{A(k)} < \mathrm{A(i+1)}, \forall \mathrm{k}, 1 \le \mathrm{k} \le \mathrm{i}$;and$\mathrm{A(k)} > \mathrm{A(i+1)}, \forall \mathrm{k}, \mathrm{j}+1 \le \mathrm{k} < \mathrm{n}$.*

There is no requirement in space. So we could use hash.

1) Try to build a hash table with a perfect hash. If there is a collision, then record the regarding index of the element a and its value v. If there is no collision, then there is no duplicate element in the array. And return "No such algorithm exists".
2) Initialize $i = a - 1, j = a + 1, p = 1, q = n$
3) While (i>p && j<q)

```
If A[p]<A[a]
        p++;
end
if A[p]==A[a]
        A[p]=A[i];
        A[i]=A[a];
        i--;
end
if A[p]>A[a]
        temp=A[p];
        A[p]=A[j];
        A[j]=temp;
        j++;
end
if A[q]>A[a]
        q--;
end
if A[q]==A[a]
        A[q]=A[j];
        A[j]=A[a];
        j++;
end
if A[q]<A[a]
        temp=A[i];
        A[i]=A[q];
        A[q]=temp;
        i--;
end
end
```

Time complexity analysis:

Step 1 needs $O(n)$ to get result. Step 2 need $O(1)$. Step 3 we move p and q at most $n-1$ times. And each time we do $O(1)$ operations. Step 3 takes $O(n)$. Thus the algorithm has a time complexity of $O(n)$.

4. *We are given an array* A *with* n *elements and a number* p. *Assume that the sum of the elements in* A *is larger than* p. *We would like to compute a smallest subset of* A' *of* A *such that the sum of the elements in* A' *is at least* p. *(For example, if* $\mathrm{A} = \{8,3,9,2,7,1,5\}$ *and* $\mathrm{p} = 18$, *then the answer is* $\mathrm{A}' = \{8,9,1\}$.*) Give an* $\mathrm{O(n)}$ *time algorithm for this problem. (**HINT**: use the linear-time SELECT algorithm)*

n is the size of the input array.

Algorithm:

1) Use linear-time SELECT algorithm to select the median of the input array.
2) Use the selected median as pivot to partition the array.
3) If the summation of the right half of the partitioned array is larger than p, then use the right half of the partitioned array as input array together with p as the threshold and run this algorithm again.
4) If the summation of the right half of the partitioned array is smaller than p. Let the summation be s. Then use the left half of the partitioned array as input array and $p - s$ as the threshold and run this algorithm again.
5) If the summation of the right half or the partitioned array is exact p, then return right half of the partitioned array.
6) If n is small enough, e.g. $n \leq 10$, use brute force to solve the question.

Time complexity analysis:

The algorithm is recursively defined. Each iteration reduces the load by half. Thus the algorithm takes $O(n)$.

5. *Let* $X = \{x_1, x_2, \cdots, x_n\}$ *be a sequence of arbitrary numbers (positive or negative). Give an* $\mathrm{O(n)}$ *time algorithm to find the subsequence of consecutive elements* $\mathrm{x_i, x_{i+1}, \cdots, x_j}$ *whose sum is maximum over all consecutive subsequence. For example, for* $\mathrm{X} = \{2,5,-10,3,12,-2,10,-7,5\}, \{3,12,-2,10\}$*is solution.*

Applying Kadane's Algorithm can solve the problem.

Algorithm implementation in Matlab:

```
function [ maxsum,maxstartindex,maxendindex ] =
maximumsub( input)
maxsum=-Inf;
maxstartindex=0;
maxendindex=0;
currentmaxsum=0;
currentstartindex=1;
for currentendindex=1:n
    currentmaxsum=currentmaxsum+input(currentendindex);
    if currentmaxsum>maxsum
        maxsum=currentmaxsum;
        maxstartindex=currentstartindex;
        maxendindex=currentendindex;
    end
    if currentmaxsum<0
        currentmaxsum=0;
```

```
            currentstartindex=currentendindex+1;
        end
    end
end
```

6. *Consider the $O(n)$ time Select Algorithm discussed in class and modify it in such a way that the group size is 9 instead of 5. Prove that the time complexity of this algorithm is also $O(n)$.*

Let $T(n)$ denote the time complexity of the Select algorithm for n elements. Note that Step 1 and 3 can be done in $O(n)$ time. Step 2 requires $T(\frac{n}{9})$ time since there are $\lfloor \frac{n}{9} \rfloor$ elements in M. Suppose we have the following:

A	A	A				
A	A	A				
A	A	A				
A	A	A				

We then make the following observations. All elements in area A are no larger than m^*; thus, at least $5\left\lceil \frac{\lfloor \frac{n}{9} \rfloor}{2} \right\rceil$ elements are less than or equal to m^*. Since $5\left\lceil \frac{\lfloor \frac{n}{9} \rfloor}{2} \right\rceil \geq 5 \cdot \frac{\lfloor \frac{n}{9} \rfloor}{2}$, we have at least $n - \frac{2.5(n-8)}{9}$ elements that are strictly larger than m^*. $\forall n \geq 80, \frac{13}{18}n + \frac{20}{9} \leq n$.

$$T(n) \leq c \cdot n + T\left(\frac{n}{9}\right) + T(\frac{3n}{4})$$

It can be shown that $T(n) \leq 30c \cdot n$.

Thus $T(n) = O(n)$.