COMP 2119A: Solution for Assignment 2

1

a) To reverse array of n numbers A[1,...,n], can call reverse A[2,...,n-1]. The detail algorithm is shown in . Time complexity: $O(n)$.

Algorithm 1 Reverse Array

```
procedure reverseArray(array A, low, high)
    if low == high then
        return A
    end if
    if high − low == 1 then
        swap(A[1ow],A[high])
        return A
    else
        reverseArray(A,low + 1,high − 1)
        swap(A[low],A[high])
    end if
end procedure
```

b) The algorithm is shown in . Time complexity: $O(\log n)$

Algorithm 2 Find element

```
procedure find(array A, low, high)
    if low == high then
        if A[low] == low then
            return true
        else
            return false
        end if
    end if
    mid = floor((low + high)/2)
    if A[mid] <= mid then
        find(A, mid, high)
    else
        find(A, low, mid-1)
    end if
end procedure
```

2

a) Use two loops to go through all situations and find the max one. The algorithm is shown in . Time complexity: $O(n^2)$.

Algorithm 3 Max Value

```
procedure maxValue(array A)
    max = 0, maxI = 0, maxJ = 0
    for i = 1, i <= n, ++i do
        for j = i, j <= n, ++j do
            if A[j] − A[i] > max then
                max = A[j] − A[i]
                maxI = i
                maxJ = j
            end if
        end for
    end for
    return max, maxI, maxJ
end procedure
```

b) If the array's size is 1, then return 0. Otherwise, we can split the array in half and find the max of left half, right half and across two halves. The algorithm is shown in . Time complexity: $O(n \log n)$.

Algorithm 4 Max Value Recursive

```
procedure maxValueMiddle(array A,low,high,middle )
    MaxRight = A[high],indexRight = high, MinLeft = A[low],IndexLeft = low
    for i = low,i <= middle,++i do
        if A[i] < MinLeft then
            MinLeft = A[i]
            IndexLeft = i
        end if
    end for
    for i = middle + 1,i <= high,++i do
        if A[i] > MaxRight then
            MaxRight = A[i]
            IndexRight = i
        end if
    end for
    return (IndexLeft,IndexRight,MaxRight − MinLeft)
end procedure
procedure maxValue(array A,low,high)
    (i1, j1, max1) = maxValue(A,low,(low + high)/2)
    (i2, j2, max2) = maxValue(A,(low + high)/2 + 1,high)
    (i3, j3, max3) = maxValueMiddle(A,low,high,(low + high)/2)
    if max1 > max2 then
        if max1 > max3 then
            return(i1,j1,max1)
        else
            return (i3,j3,max3)
        end if
    else
        if max2 > max3 then
            return (i2,j2,max2)
        else
            return (i3,j3,max3)
        end if
    end if
end procedure
```

3

Firstly, move n red disks from 0 to 2(using 0,1,2). Then, move n blue disks from 3 to 0 (using 0,1,3). Finally, move n red disks from 2 to 3 (using 1,2,3). The algorithm is shown in .

Algorithm 5 TofH

procedure TofH(A,B,C,n)
    if $n = 1$ **then**
        move disk from A to C
    else
        TofH(A,C,B,$n-1$)
        move disk from A to C
        TofH(B,A,C,$n-1$)
    end if
end procedure
procedure TofHdoulbe(nR,nB,0,1,2,3)
    TofH(0,1,2,nR)
    TofH(3,1,0,nB)
    TofH(2,1,3,nR)
end procedure

4

Assume that we have a stack S, we define a new stack S' whose elements are like $(a1, a2)$ where $a1, a2 \in S$. The algorithm is shown in , while the Top and Pop are same as usual stack. The time complexity of four operations are $O(1)$.

Algorithm 6 Stack

procedure Push(S,a)
    $x = Top(S)$
    if $x > a$ **then**
        $S'.push((a, a))$
    else
        $S'.push((a, x))$
    end if
end procedure
procedure FindMin(S)
    $x = S'.Top()$
    return $x.a1$
end procedure

5

a) Allocate a $n * n$ matrix M, whose entries are set as 0. For any two vertex i and j, $M[i][j]$ equals to number of edges pointing from i to j. For vertex i: in-degree is sum of the ith row in M, and out-degree is the sum of the ith column. The time complexity is $O(n)$, since traverse one row or one column.

b) We can check $M[i][j], M^2[i][j], ..., M^n[i][j]$. If there exist $M^k[i][j] > 0$, then j can be reached from i in the graph. Or you can use DFS or BFS to check if there is a path between j and i. The time complexity of DFS or BFS is $O(n^2)$.

c) We can use BFS with some starting vertex, e.g. $s = 1$. If the number of accessed vertexes during the search equals to n, then it is connected, else not. The algorithm is shown in . Time complexity is

$O(n^2)$.

Algorithm 7 BFS

```
procedure BFS(M,s)
    queue Q
    visited[0,...,n-1]=false
    count=0
    visited[s] = true
    Enqueue(Q,s)
    while not(Empty(Q)) do
        s=Dequeue(Q)
        count++
        for i = 0,i < n,++i do
            if !visited[i] and i ≠ s and M[i][s] > 0 then
                Enqueue(Q,i)
                visited[i]=true
            end if
        end for
    end while
    if count == n then
        return true
    elsereturn false
    end if
end procedure
```

d) We can use DFS to travel the graph, and keep a record of the parent node of node i. The algorithm is shown in . Time complexity is $O(n^2)$

Algorithm 8 DFS

```
procedure DFS(M,s,pre)
    visit[s] = true
    for i = 0,i <= n,++i do
        if M[s][i] == 1 then
            if !visit[i] then
                return DFS(i,s)
            else
                if i! = pre then
                    return false
                end if
            end if
        end if
    end for
    return true
end procedure
```

6

a)

Algorithm 9 EnhancedBFS

```
procedure EnhancedBFS(s,n)
    queue Q
    int visited[1, ··· , n − 1] = {0, ··· , 0}
    Enqueue (Q,s)
    visited[s] = 1
    while Q not empty do
        i=Dequeue(Q)
        for each neighbour j of i do
            if visited[j]=0 then
                Enqueue(Q,j)
                visited[j]=1
                PREV[j]=i
            end if
        end for
    end while
end procedure
```

b)

Algorithm 10 printv

```
procedure Print(v)
    if v!=s then
        if PREV[v]!=s then
            print(PREV[v])
            print v
        end if
    end if
end procedure
```

7

a)simple weighted graph (ABCD||Φ)
b)each node represents a state of the situation(where the four people are),|| denotes the river.
c)edges are the time required to move from one node to the next node
d)find the minimal weighted path in weighted directed graph from the original node (ABCD||Φ) to final state (Φ||ABCD)

8

	0	1	2	3	4	5	6	7	8	9	10
Linear probing	22	88			4	15	28	17	59	31	10
quadratic probing	22		88	17	4		28	59	15	31	10
double hashing	22		59	17	4	15	28	88		31	10

9

Proof: Assume that we have $0 \le i, j \le m-1, i \le j$, such tnat $\frac{1}{2}i(i+1) \mod m = \frac{1}{2}j(j+1) \mod m$.

That is to say, $\exists k_1, k_2 \in Z$, we have $k_1 m + \frac{1}{2}i(i+1) = k_2 m + \frac{1}{2}j(j+1)$.

Then, we can get that $2(k_1 - k_2)m = i^2 + i - j^2 - j = (j-i)(j+i+1)$.

1) When both i, j are even or odd, then $i+j+1$ is odd. Since $m = 2^p$, so we have $gcd(2m, i+j+1) = 1$. So, $j - i = k' * 2m, k' \in Z$. Since $j - i \le m - 1 < 2m$, so $k' = 0$. So we have $i = j$.

2) When one of i, j is even and the other is odd, then $j - i$ is odd. Since $m = s^p$, so we have $gcd(2m, j-i) = 1$. So, $i + j + 1 = t' * 2m, t' \in Z$. Since $1 \le i + j + 1 < 2m - 1$, so no such t' exists. Therefore, we cannot pick one add and one even i, j such that $(j-i)(i+j+1) = 2(k_1 - k_2)m$.

We can get that if $h(k,i) = h(k,j)$, then $i = j$. So, the probe sequence $< h(k,0), h(k,1), ..., h(k,m-1) >$ is a permutation of $< 0, 1, 2, ..., m-1 >$.