Algorithms Pseudo Code

BUE Final Exam 2023 (**IMPORTANT**)

The chess Rook can move horizontally or vertically any number of squares in the chessboard as long as its path is empty. Use dynamic programming to find the number of shortest paths by which a Rook can move from one corner of a chessboard to the diagonally opposite corner. The length of a path is measured by the number of squares it passes through including the first and the last squares. [10

```
ALGORITHM find_unique_paths(x,y)
    //to find all unique paths of rook using Dynamic Programing
    //Input: start point x, y
    //Output: number of paths
    if x = n and y = n
        return 1
    if dp[x][y] ≠ 0
        return dp[x][y]
    if x = n
        dp[x][y] ← find_unique_paths(x, y+1)
    else if y = n
        dp[x][y] ← find_unique_paths(x+1, y)
    else
        dp[x][y] ← find_unique_paths(x+1, y) + find_unique_paths(x, y+1)
    return dp[x][y]
```

Write a pseudocode for a divide-and-conquer algorithm for finding the smallest element in an $n \times n$ matrix. Then, compute its order of growth. n^2

```
ALGORITHM findSmallest(matrix,rowStart,rowEnd, colStart,colEnd)

    //find smallest element in matrix of size n*n

    //Input: matrix, row start, row end, column start and column end

    //Output: minimum element

    if rowStart = rowEnd and colStart = colEnd)

        return matrix[rowStart][colStart]


    rm ← floor((rowStart+rowEnd)/2)

    cm ← floor((colStart+colEnd)/2)

    if colStart = colEnd

        return min(findSmallest(matrix,rowStart,rm,colStart,colEnd),findSmallest(matrix,rm+1,rowEnd,colStart,colEnd))

    if rowStart = rowEnd

        return min(findSmallest(matrix,rowStart,rowEnd,colStart,cm),findSmallest(matrix,rowStart,rowEnd,cm+1,colEnd))

    return min(min(findSmallest(matrix,rowStart,rm,colStart,cm), findSmallest(matrix,rowStart,rm,cm+1,colEnd))

                    ,min(findSmallest(matrix,rm+1,rowEnd,colStart,cm),findSmallest(matrix,rm+1,rowEnd,cm+1,colEnd)))
```

Algorithms Pseudo Code

.2 [Total: 36 marks]

Consider the job assignment problem with n persons and m jobs where $n \leq m$. Assume also if we assign person p to job j it will cost c_{pj}. (n m)

i. Design a greedy algorithm in the form of pseudo-code that finds the optimal assignment that minimizes the total cost assuming the odd numbered jobs can be assigned only to the odd numbered persons. [9 marks]

```
ALGORITHM min_cost(jobs[0....m-1],n,m,cpi[0.....n-1][0.......m-1])
    //minimize total cost of assign odd n jobs to odd n persons
    //Input: jobs, number of persons n, number of jobs m, and cpi cost of every job
    //Output: minimum total cost
   //O(nm)
    total_cost ← 0
    for j ← 0 to n-1 do
      for i ← 0 to m-1 do
        if i % 2 = 0 || i % 2 = j % 2
            jobs[i] ← min(jobs[i],cpi[j][i])
    for i ← 0 to m-1 do
        total_cost ← total_cost + jobs[i]
   return total_cost
```

Consider the job assignment problem with n persons and m jobs where $n \leq m$. Assume also if we assign person p to job j it will cost c_{pj}, and we need to find the optimal assignment that minimizes the overall cost. Design a dynamic programming algorithm to solve this problem, then compute its order of growth. [10 ma

```
ALGORITHM job_assignment(C[0...n-1][0....m-1],dp[0....1<<m-1],n,m)
    //assign m jobs to n persons using dynamic programming
    //Input: cost matrix C[0...n-1][0....m-1], array dp[0....1<<m-1],n, m;
    //Output: return minimum cost
    dp[0] ← 0
    for msk ← 1 to 1<<m-1 do
        dp[msk] ← INT_MAX
        for j ← 0 to m-1 do
            if  msk & (1 << j)
                prev_msk ← msk & ~(1<<j)
                cost ← C[__builtin_popcount(prev_msk)][j]
                dp[msk] ← min(dp[msk], dp[prev_msk] + cost)
    return dp[(1<<m)-1]
```

Algorithms Pseudo Code

Question 2: **(8 marks)**

Consider the job assignment problem with *n* persons and *m* jobs where $n < m$. Assume also if we assig person *p* to job *j* it will cost c_{pj}. Design a brute-force algorithm in the form of pseudo-code that finds th optimal assignment that minimizes the total cost so that the odd numbered jobs can be assigned only to th odd numbered persons. Then, compute its order of growth.

```
ALGORITHM solve(st,noOfJobs,currSum,v[0....n-1][0....m-1])
    //assign m jobs to n persons using brute force odd number jobs to odd persons
    //Input: v[0....n-1][0....m-1] cost matrix, number of persons and jobs
    //Output: minimum cost
    if noOfJobs = m
        minCost ← min(minCost,currSum)
        return
    for i ← 0 to n-1 do
        for j ← st to m-1 do
            if taken[j]
                continue
            if j%2 = 1 and i%2 ≠ 1 //odd to odd
                continue
            else
                taken[j] ← 1
                solve(st,noOfJobs+1,currSum+v[i][j])
                taken[j] ← 0
                solve(st+1,noOfJobs,currSum)
```

(4 marks)

Consider the job assignment problem with *n* persons and *m* jobs where $n \le m$. Assume also if we assign person *p* to job *j* it will cost C_{pj}. Assume that the even-numbered jobs can be assigned only to the even-numbered persons. Design a brute-force algorithm in the form of pseudo-code that finds the optimal assignment that minimizes the total cost. Then, compute its order of growth in Big-O notation. **(8 marks)**

Assume we have a set of symbols X, Y, Z and W that ...

```
ALGORITHM solve(st,noOfJobs,currSum,v[0....n-1][0....m-1],n,m)
    //assign m jobs to n persons using brute force even number jobs to even persons
    //Input: v[0....n-1][0....m-1] cost matrix, number of persons and jobs
    //Output: minimum cost
    if noOfJobs = m
        minCost ← min(minCost,currSum)
        return
    for i ← 0 to n-1 do
        for j ← st to m-1 do
            if taken[j]
                continue
            if j%2 ≠ 1 and i%2 = 1 //even to even
                continue
            else
```

```
                taken[j] ← 1
                solve(st,noOfJobs+1,currSum+v[i][j])
                taken[j] ← 0
                solve(st+1,noOfJobs,currSum)
```

Question 3: (8 marks)

A. Consider the job assignment problem with n persons and m jobs where $n \le m$. Assume also if we assign person p to job j it will cost c_{pj}. Assume that the even numbered jobs can be assigned only to the odd numbered persons: **(8 marks)**

i. Design a brute-force algorithm in the form of pseudo-code that finds the optimal assignment that minimizes the total cost.

ii. Compute the order of growth for the algorithm in (i).

```
ALGORITHM solve(st,noOfJobs,currSum,v[0....n-1][0....m-1],n,m)
    //assign m jobs to n persons using brute force even number jobs to odd persons
    //Input: v[0....n-1][0....m-1] cost matrix, number of persons and jobs
    //Output: minimum cost
    if noOfJobs = m
        minCost ← min(minCost,currSum)
        return
    for i ← 0 to n-1 do
        for j ← st to m-1 do
            if taken[j]
                continue
            if j%2 ≠ 1 and i%2 ≠ 1 //even to odd
                continue
            else
                taken[j] ← 1
                solve(st,noOfJobs+1,currSum+v[i][j])
                taken[j] ← 0
                solve(st+1,noOfJobs,currSum)
```

Final Spring 2022

You have a row of 4n disks of two colors, 2n dark and 2n light. They alternate dark, light, dark, light, and so on. You want to get all the dark disks to the right-hand end, and all the light disks to the left-hand end. The only moves you are allowed to make are those that interchange the positions of two neighboring disks. Design a divide-and-conquer algorithm in the form of pseudo-code for solving this problem and determine its efficiency.

```
ALGORITHM merge_disks(disks,start,mid,end)
    //merge disks
    //Input: disks array, start, mid, and end
```

```
    //Output: modified disks
    i ← start
    j ← mid+1
    k ← 0
    while i ≤ mid and j ≤ end do
        if  disks[i] = 'D' and disks[j] = 'L'
            sorted_disks[k] ← 'L'
            k ← k + 1
            sorted_disks[k] ← 'D'
            k ← k + 1
            i ← i + 1
            j ← j + 1
        else
            sorted_disks[k] ← disks[i]
            k ← k + 1
            i ← i + 1
            sorted_disks[k] ← disks[j]
            k ← k + 1
            j ← j + 1

    while i ≤ mid do
        sorted_disks[k] ← disks[i]
        k ← k + 1
        i ← i + 1

    while j ≤ end do
        sorted_disks[k] ← disks[j]
         k ← k + 1
         j ← j + 1

    for  x = 0 to  k do
        disks[start+x] ← sorted_disks[x]



ALGORITHM sort_disks(disks,start,end)
     //sort disks
    //Input: disks array, start, and end
    //Output: sorted disks
    if start < end
        mid ← (start + end) / 2
        sort_disks(disks, start, mid)
        sort_disks(disks, mid+1, end)
        merge_disks(disks, start, mid, end)
```

Algorithms Pseudo Code

Consider an arrangement of integers from 1 to n^2 in an $n \times n$ matrix, with each number occurs exactly once so that the sum of the elements in each row is the same, which is equal to the sum of the elements in each column, and also equal to the sum of the elements in each main diagonal. Design an exhaustive search algorithm in the form of pseudo-code for generating all $n \times n$ matrices that satisfy the condition above. Then, compute the Big-O for the designed algorithm.

ملحوظه دي لعنه
يارب ما تيجي

```
// check whether the matrix is valid
ALGORITHM is_valid(matrix[0....n-1][0....n-1],n)
    //check valid magic or not
    //Input: matrix[0....n-1][0....n-1] and size n
    //Output: true or false is magic or not
    for i ← 0 to n-1 do
        row_sum ← 0
        for j ← 0 to n-1 do
            row_sum ← row_sum + matrix[i][j]
        if row_sum ≠ sum
            return false

    for j ← 0 to n-1 do
        col_sum ← 0
        for i ← 0 to n-1 do
            col_sum ← col_sum + matrix[i][j]

        if col_sum ≠ sum
            return false

    main_diag_sum ← 0
    for i ← 0 to n-1 do
        main_diag_sum ← main_diag_sum + matrix[i][i]

    if main_diag_sum ≠ sum
        return false

    sec_diag_sum ← 0
    for i ← 0 to n-1 do
        sec_diag_sum ← sec_diag_sum + matrix[i][n-i-1]
    if sec_diag_sum ≠ sum
        return false
    return true
```

Algorithms Pseudo Code

```
// generate all permutations of the remaining unused numbers and try to fill in the matrix
ALGORITHM backtrack(index)
    //generate all permutations and save the solution which achieve that valid magic square
    //Input: index
    //Output: all possible magic square are saved in solutions (vector of vectors)
    if index = n*n
        if is_valid(matrix,n)
            for  i ← 0 to n-1 do
                for j ← 0 to n-1 do
                    sol.push_back(matrix[i][j])
            solutions.push_back(sol)
        return

    for num ← 1 to n*n do
        if not(used[num])
            i ← index / n
            j ← index % n
            matrix[i][j] ← num
            used[num] ← true
            backtrack(index+1)

            used[num] ← false
```

Algorithms Pseudo Code

Consider a list of 3*n* shapes that consists of *n* triangles, *n* circles, and *n* squares as shown in the below figure. The shapes are randomly ordered in the list. We need to move all the circles in the list to be on the left-hand side of the list, the squares in the right-hand side, and the triangles to the middle of the list. The only moves you are allowed to make are those that interchange the positions of two neighbouring shapes. Design a brute-force algorithm for solving this problem (write your algorithm in the form of pseudo-code and provide your assumptions) and determine the number of moves it takes. Then, compute Big-O for the designed algorithm.

```
ALGORITHM sort_shapes(shapes[0.....3*n-1],n)

    //sort shapes to circle then triangle then square
    //Input: array of shapes and size n
    //Output: sorted shapes
    for i ← 1 to 3*n-1 do
        if shapes[i] = "triangle" and shapes[i-1] = "square"
            t1 ← i-1
            while shapes[t1] = "square" and t1 ≥ 0){
                swap(shapes[t1],shapes[t1+1]);
                t1 ← t1 - 1

        else
            if shapes[i] = "circle" and shapes[i-1] = "triangle" or
               shapes[i] = "circle" and shapes[i-1] = "square"
                t2 ← i-1
                while shapes[t2] ≠ "circle" and t2 ≥ 0 do
                    swap(shapes[t2],shapes[t2+1])
                    t2 ← t2 - 1
```

Fall 2020

Write a pseudocode for a divide-and-conquer algorithm for finding the largest element in an array of ***n*** numbers. Then, compute its order of growth. **(5 marks)**

```
ALGORITHM largest(arr,low,high)

    //get largest element in array using divide and conquer technique
    //Input: array of elements, low points to first index, and high points to last index
    //Output: largest element
    if low = high
        return arr[low]
    else
        mid ← (low+high)/2
        leftMax ← largest(arr,low,mid)
        rightMax ← largest(arr,mid+1,high)
        return max(leftMax,rightMax)
```

Algorithms Pseudo Code

Spring 2019

Assume we need to count the number of substrings that start with letter **A** and end with letter **M** in a given text. For example, there are two such substrings in the text **AIN_SHAMS_UNIVERSITY**, which are **AIN_SHAM** and **AM**. **(4 marks)**

i. Design a brute-force algorithm to find this number in any arbitrary string (write your algorithm in the form of pseudocode and state your assumptions).
ii. Compute the worst-case order of growth of the designed algorithm in (i).

```
ALGORITHM match(s)

    //get the number of Substrings in the given string start by A and end by M
    //Input: given string s
    //Output: number of Substrings
    n ← s.length()
    count ← 0;
    for i ← 0  to n-2 do
        if s[i] = 'A'
            for j ← i+1 to n-1 do
                if s[j] = 'M'
                    count ← count + 1
    return count
```

Summer 2019

You have a row of 2*n* binary digits, *n* zeros and *n* ones. They alternate: zero, one, zero, one, and so on. You want to get all the zeros to the right-hand end, and all the ones to the left-hand end. The only moves you are allowed to make are those that interchange the positions of two neighboring digits. **(4 marks)**

i. Design a brute-force algorithm for solving this problem, write your algorithm in pseudo-code form.
ii. Determine the number of moves it takes to get all the zeros to the right-hand end, and all the ones to the left-hand end.
iii. Find the big-O for the designed algorithm.

```
ALGORITHM sort_digits(digits[0.....2*n-1],n)
    //sort shapes to circle then triangle then square
    //Input: array of shapes and size n
    for i ← 1 to 2*n-1 do
        if digits[i] = 1 and digits[i-1] = 0
            t1 ← i-1
            while digits[t1] = 0 and t1 ≥ 0)
                swap(digits[t1],digits[t1+1])
                t1 ← t1 - 1
```

Algorithms Pseudo Code

Assume we have *n* positive integers and we need to partition them into two disjoint subsets with the same sum (as possible) of their elements. Design an exhaustive-search algorithm (in the form of pseudo-code) for this problem taking into consideration to minimize the number of subsets the algorithm needs to generate. Then, design a greedy algorithm for this problem and find its order of growth. Discuss if the designed greedy solution always finds the correct partitioning. Justify your answer.

```
//greedy
#include <bits/stdc++.h>
using namespace std;

//
ALGORITHM subset (arr,n,sum)
   //check there are subset using greedy
   //Input: array of elements, n is size, sum from partition ALGORITHM
   //Output: true or false sum goes to zero or not
   sort(arr, arr + n, greater<int>())
   i ← 0
   while sum > 0 and i < n do
      if arr[i] ≤ sum
         sum ← sum - arr[i]
      i ← i + 1
   return sum = 0


ALGORITHM partition (arr, n)
    //check array can partitioned
    //Input: array of elements, n size
    //Output: true or false can partitioned
   sum ← 0
   for i ← 0 to n-1 do
      sum ← sum + arr[i]
   // If sum is odd, there cannot be two subsets
   // with equal sum
   if sum % 2 ≠ 0
      return false
   return subset (arr, n, sum/2)

//brute force

ALGORITHM subset (arr,n,sum)
    //check there are subset using exhaustive search
   //Input: array of elements, n is size, sum from partition ALGORITHM
   //Output: true or false sum goes to zero or not
```

Algorithms Pseudo Code

```
   if  sum = 0
        return true
   if  n = 0 and sum ≠ 0
        return false

   // If last element is greater than sum, then ignore it
   if  arr[n-1] > sum
        return subset (arr, n-1, sum)
   // check if sum can be obtained by excluding the element or including it
   return subset (arr, n-1, sum) || subset (arr, n-1, sum-arr[n-1])


ALGORITHM partition (arr, n)
    //check array can partitioned
    //Input: array of elements, n size
    //Output: true or false can partitioned
   sum ← 0
   for i ← 0 to n-1 do
      sum ← sum + arr[i]
   // If sum is odd, there cannot be two subsets
   // with equal sum
   if sum % 2 ≠ 0
      return false
   return subset (arr, n, sum/2)
```

c. Consider the problem of scheduling n jobs of known durations $t_1, t_2 \ldots t_n$ for execution by a computing machine with m processors, where $m \leq n$. The jobs can be executed in any order on any available processor that has currently no assigned job, but once the job started on a given processor, the processor will not be relinquished until it finishes executing the assigned job. We need to find a schedule that minimizes the total time spent by all the jobs in the system. Assume that the time spent by one job in the system is the sum of the time spent by this job in waiting plus the time spent on its execution.

i. Formulate the objective function and the constraints of this optimization problem in mathematical form. **(1 mark)**

ii. Design a greedy algorithm to find the required schedule for this problem (write your algorithm in the form of pseudocode and state your assumptions). Indicate if this algorithm always yields an optimal solution, justify your answer. **(2 marks)**

Algorithms Pseudo Code

```
ALGORITHM solve(s,pn)
    //Input: vector s, number of processes
    //Output: total time spent by all processes
    sort(s.begin(),s.end())
    ans ← 0
    for i ← 0 to s.size() do
        v ← INT32_MAX
        pos ← 0
        for j ← 0 to pn-1 do
            if p[j]+s[i] < v do
                v ← p[j]+s[i]
                pos ← j
        jdone[i] ← v
        p[pos] ← v
        ans ← ans + v
    return ans;
```

The chess Rook can move horizontally or vertically any number of squares in the chessboard as long as its path is empty. Design a greedy algorithm in the form of pseudo-code to find the number of paths by which a Rook can move from one corner of a chessboard to the diagonally opposite corner. The length of a path is measured by the number of squares it passes through including the first and the last squares. Then, find the efficiency of the designed algorithm.

```
      ALGORITHM dfs (x,y,last_step)

    //Input: x,y, last_step
    //Output: all number of paths
    if last_step = 'R'
        for  i ← x+1  to N-1 do
            dfs (i,  y,  'D')

    else if  last_step = 'D'
        for i ← y+1 to N-1 do
            dfs (x,  i,  'R')
    else
        for i ← 1 to N-1 do
            dfs (i,  y,  'D')
        for i ← 1 to N-1 do
            dfs (x,  i,  'R')

    if x = N -1 and y = N-1
        numberOfways ← numberOfways + 1
```

Algorithms Pseudo Code

Consider the problem of scheduling n jobs of known durations $t_1, t_2 \ldots t_n$ for execution by a computing machine with m processors, where $m \leq n$. The jobs can be executed in any order on any available processor that has currently no assigned job, but once the job started on a given processor, the processor will not be relinquished until it finishes executing the assigned job. We need to find a schedule that minimizes the total time spent by all the jobs in the system. Assume that the time spent by one job in the system is the sum of the time spent by this job in waiting plus the time spent on its execution. Design a dynamic programming algorithm in the form of pseudo-code to find the optimal schedule, show the detailed steps you used and state your assumptions.

العنوان /

```
int find_min_time(int sched[][], int waiting_queue[]
                  , int n, m, bool run_state)
{   int i = 0;
    int dp[n][m];
    do{
        wait_queue[i]++;
        i = (i+1) % wait_que.size();
    }while(run_state)

    for (int i=0; i<n; i++){
      for(int j=0; j<m; j++){
         dp[i][j] = sched[i][j];
      }
    }

    for (int i=0; i<n; i++) {
      for(int j=0; j<m; j++){
        if(dp[i][j] != 0)
          for(int k=0; k<m; k++){
            if( waiting_que[k] != 0)
            {
              dp[i+1][j] = min(dp[i][j],
                               dp[i+1][j] + waiting_que[k])
              sched[i][k] = 0;
            }
          }
      }
    }
    return max (dp[n][m]);
}
```