Introduction to Algorithms Lecture 3

By: Darmen Kariboz

Chess and 8 queens

Arrange all 8 queens so that no queens will attack each other.

8 queens

```
1.  Function arrangeQueens(map, row)
2.      if row > 8 then
3.          return true
4.      For col = 1 to 8 do
5.          if map[ row ] [ col ] = 0 then
6.              increase8Directions(map,
    row, col )
7.              arrangeQueens(map, row + 1)
8.              decrease8Directions(map,
    row, col )
```

Breadth-first search

Find exit from labyrinth.

Sample input:

```
6
1 1 1 1 1 1
1 0 2 1 0 0
1 0 1 1 0 1
1 0 0 0 0 1
1 0 1 0 0 1
1 1 1 0 1 1
```

Sample output:

```
1  1  1  1  1  1
1  3  2  1  10 11
1  4  1  1  9  1
1  5  6  7  8  1
1  6  1  8  9  1
1  1  1  9  1  1
```

BFS – Breadth-first Search

```
1.  loadMap(map)
2.  Q ← (x,y) //enque first position
3.  k = 2
4.  map[x][y] = k
5.  while size(Q) > 0 do
6.      foreach (x,y) ← Q do // deque from queue
7.          if map[x+1][y] = 0 then // check North
8.              map[x+1][y] = k+1
9.              T ← (x+1,y)
10.         if …. // repeat for each side, N,E,W,S.
11.     k = k+1
12.     Q = T
```

Depth-first search

Monkey and bananas.

Sample input:	Sample output:
6 5 2 3 5 7 3 1 3 5 6 2 3 4 7 3 3 6 4 1 3 5	30

DFS – Depth-first Search

```
1. Function findMax( treeArray, X, Y, sum)
2.     if X > rowCount(treeArray) then
3.         if max < sum then
4.             max = sum
5.         return
6.     findMax( treeArray, X+1, Y,
   sum+treeArray[X][Y])
7.     findMax( treeArray, X+1, Y+1,
   sum+treeArray[X][Y])

8. // in main call
9. findMax( treeArray, 1, 1, 0)
```

$$P_k^n = \frac{n!}{(n-k)!}$$

Permutation

Permute all anagrams of given word.

Permutation

```
1.      Function permute( array, index)
2.          if index = length(array)
3.              for j=0 to length(array) do
4.                  output array[j]
5.          else
6.              for i=index to length(array) do
7.                  array = a[index] <-> a[i]
8.                  permute(array, index+1)
9.                  array = a[index] <-> a[i]
```

Quick Sort

Select **pivot**, put all elements smaller than **pivot** to left part, and move elements greater than **pivot** to right part.

Then repeat this process for each part.

Quick sort (Partition)

```
1.  function partition(array, left, right, pivotIndex)
2.        pivotValue := array[pivotIndex]
3.        swap array[pivotIndex] and array[right] // Move pivot to end
4.        storeIndex := left
5.        for i from left to right - 1 // left ≤ i < right
6.            if array[i] ≤ pivotValue
7.                swap array[i] and array[storeIndex]
8.                storeIndex := storeIndex + 1
9.        swap array[storeIndex] and array[right] // Move pivot to its final place
10.       return storeIndex
```

Quick Sort (quicksort)

```
1. procedure quicksort(array, left, right)
2.     if right > left
3.         select a pivot index //(e.g. pivotIndex := left + (right - left)/2)
4.         pivotNewIndex := partition(array, left, right, pivotIndex)
5.         quicksort(array, left, pivotNewIndex - 1)
6.         quicksort(array, pivotNewIndex + 1, right)
```

Merge Sort

Separate list into two equal halves, Execute **Merge sort** for each of them, then join two list into one sorted array.

Merge Sort (merge_sort)

```
1.  function merge_sort(m)
2.      if length(m) ≤ 1
3.          return m
4.      var list left, right, result
5.      var integer middle = length(m) / 2
6.      for each x in m up to middle
7.          add x to left
8.      for each x in m after middle
9.          add x to right
10.     left = merge_sort(left)
11.     right = merge_sort(right)
12.     result = merge(left, right)
13.     return result
```

Merge Sort (merge)

```
1.  function merge(left,right)
2.      var list result
3.      while length(left) > 0 or length(right) > 0
4.          if length(left) > 0 and length(right) > 0
5.              if first(left) ≤ first(right)
6.                  append first(left) to result
7.                  left = rest(left)
8.              else
9.                  append first(right) to result
10.                 right = rest(right)
11.         else if length(left) > 0
12.             append first(left) to result
13.             left = rest(left)
14.         else if length(right) > 0
15.             append first(right) to result
16.             right = rest(right)
17.     end while
18.     return result
```

Home Work

- **A+B=C**
- Given three numbers: A, B, C. Find all possible ways to, change digits in A and B, so that sum will be C.

- **LABYRINTH**
- Find all possible ways to exit from labyrinth, by use of DFS.

- **QUICK SORT (Groups: A, C, E, G)**
- Realize quick sort for any numbers given in console.

- **MERGE SORT (Groups: B, D, F, H)**
- Realize merge sort for any numbers given in console.