Recitation 3 (Recursion + time complexity)

Question 1: A palindrome is a sequence of characters or numbers that looks the same forwards and backwards. For example, "**dennis sinned**" is a palindrome because it is spelled the same reading it from front to back as from back to front. The number **12321** is a numerical palindrome. Write a function that takes a string and its length as arguments and recursively determines whether the string is a palindrome:

```
bool isPalindrome(const char *str, const int len){
     if (len <= 1)
      return true;

     return ((str[0] == str[len - 1]) && isPalindrome(str + 1, len - 2));
}
```

What is the time complexity of this function in terms of Big-O notation? *Answer: O(len)*

Question 2: (Old midterm question) Consider the problem of finding the k element subsets of a set with n elements. Write a recursive function that takes an array of integers representing the set, the number of integers in the set (n), and the required subset size (k) as input, and displays all subsets with k elements on the screen. You may assume that the elements in the array have unique values.

For example, if the array (set) contains the elements [8 2 6 7], n is 4, and k is 2, then the output is

82
86
87
26
27
67

```
void findSubsets(const int *array, const int n, const int k,
                 int *taken, const int i) {
     if (k == 0){
            for (int j = 0; j < n; j++)
                  if (taken[j])
                        cout << array[j];
            cout << endl;
     }
     else if (i < n) {
            taken[i] = 1;
            findSubsets(array,n,k - 1,taken,i + 1);
            taken[i] = 0;

            findSubsets(array,n,k,taken,i + 1);
     }
}
// this is the function to be called
void findSubsets(const int *array, const int n, const int k){
     int *taken = new int[n];
     for (int i = 0; i < n; i++)
            taken[i] = 0;

     findSubsets(array,n,k,taken,0);
     delete []taken;
}
int main(){
     int a[] = {8, 2, 6, 7};

     findSubsets(a,4,2);
     return 0;
}
```

Question 3: (Old midterm question) Consider a salesman at a grocery store. The salesman uses a scale with two buckets to weigh the purchased items. The salesman also has a finite number of metal weights (e.g., 1kg, 4kg, 8kg, 9kg) to be used in the weighing process. Write a recursive function that checks whether these metal weights are sufficient to weigh a purchased item.

- Your recursive function should take the available metal weights and the weight of the purchased item as inputs. The function returns true if the metal weights can be used to weigh the item; returns false otherwise.
- For example, if the available metal weights are 1kg, 4kg, 8kg, and 9kg, and the purchased item is 10kg, the salesman can put this item into one bucket and the 1kg and 9kg metal weights into the other bucket to weigh this item. If the purchased item is 16kg, the salesman can put this item and the 1kg metal weight into one bucket, and the 8kg and 9kg weights into the other bucket to weigh this item. However, an 19kg item cannot be measured using these metal weights.

```
bool scale(const int *weights, const int size, const int item,
           const int i, const int sum) {

     if (sum == item)
            return true;

     if (i == size)
            return false;

     bool result = false;

     result = scale(weights, size, item, i + 1, sum + weights[i]);

     if (result == false)
            result = scale(weights, size, item, i + 1, sum - weights[i]);

     if (result == false)
            result = scale(weights, size, item, i + 1, sum );

     return result;
}

// this is the function to be called
bool scale(const int *weights, const int size, const int item){

     return scale(weights,size,item,0,0);
}

int main(){
     int W[] = {1, 4, 8, 9};

     cout << scale(W,4,10) << endl;
     cout << scale(W,4,16) << endl;
     cout << scale(W,4,19) << endl;

     return 0;
}
```

Question 4: Now let's modify the previous question so that it will display how we need to use the weights on the scale. For example,

If the purchased item is 10kg, then the output should be

Put the purchased item on the left bucket
Put 1kg, 9kg on the right bucket

If the purchased item is 16kg, then the output should be

Put the purchased item on the left bucket
Put 1kg on the left bucket
Put 8kg, 9kg on the right scale

If the purchased item is 19kg, then the output should be

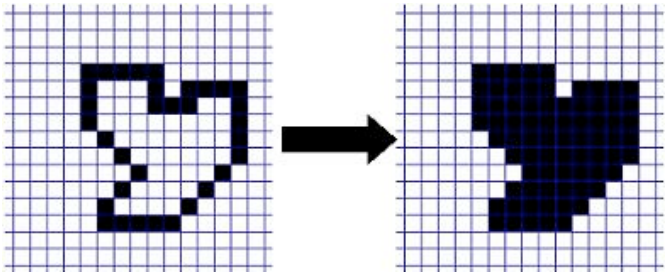
It is not possible to weigh this purchased item

```
bool scale(const int *weights, const int size, const int item,
           const int i, const int sum, int *used) {

      if ( sum == item )
            return true;

      if ( i == size )
            return false;

      bool result = false;

      result = scale(weights, size, item, i + 1, sum + weights[i], used);
      used[i] = 1;

      if ( result == false ){
            result = scale(weights, size, item, i + 1, sum - weights[i], used);
            used[i] = -1;
      }

      if ( result == false ) {
            result = scale(weights, size, item, i + 1, sum, used);
            used[i] = 0;
      }
      return result;
}

// This is the function to be called
void scale(const int *weights, const int size, const int item){
      int *used = new int[size];

      for (int i = 0; i < size; i++)
            used[i] = 0;

      if (scale(weights,size,item,0,0,used)){

            cout << "Put the purchased item on the left bucket" << endl;

            // Check the metal weights that are to be put on the left scale
            // The first flag is used for formatting purposes. It is also used
            // whether or not there is at least one item to be put on
            // the left scale
            bool first = true;

            for (int j = 0; j < size; j++){
                  if (used[j] == -1){
                        if (first){
                              cout << "Put " << weights[j] << "kg";
```

```
                              first = false;
                        }
                        else
                              cout << ", " << weights[j] << "kg";
                  }
            }
            if (first == false)
                  cout << " on the left bucket" << endl;


            // Now check the metal weights that are to be put on the right scale
            first = true;
            for (int j = 0; j < size; j++){
                  if (used[j] == 1){
                        if (first){
                              cout << "Put " << weights[j] << "kg";
                              first = false;
                        }
                        else
                              cout << ", " << weights[j] << "kg";
                  }
            }
            if (first == false)
                  cout << " on the right bucket" << endl;

      }
      else
            cout << "It is not possible to weigh the purchased item" << endl;

      delete []used;
}

int main(){
      int W[] = {1, 4, 8, 9};

      scale(W,4,10);
      scale(W,4,16);
      scale(W,4,19);

      return 0;
}
```

Question 5: Suppose that we will implement a flood fill function for black-and-white images, in which black and white are represented with 0 and 1, respectively. This function takes a 2D integer array, its dimensions, and the x and y coordinates of a pixel. Then, starting from this pixel, it recursively changes the white pixels that are adjacent to the pixel of interest to black, given that this pixel of interest is white in the original image. Assume that the function uses 4-adjaceny. Note that if the starting pixel is black in the original image, the function will do nothing.

Below is an example of an image before and after calling this function.



```
void floodFill(int **image, const int dimx, const int dimy,
               const int x, const int y, int **visited){

      if (image[x][y]){
            image[x][y] = 0;

            if ((x - 1 >= 0) && !visited[x - 1][y] && image[x - 1][y]){
                  visited[x - 1][y] = 1;
                  floodFill(image,dimx,dimy,x - 1,y,visited);
            }
            if ((x + 1 < dimx) && !visited[x + 1][y] && image[x + 1][y]){
                  visited[x + 1][y] = 1;
                  floodFill(image,dimx,dimy,x + 1,y,visited);
            }
            if ((y - 1 >= 0) && !visited[x][y - 1] && image[x][y - 1]){
                  visited[x][y - 1] = 1;
                  floodFill(image,dimx,dimy,x,y - 1,visited);
            }
            if ((y + 1 < dimy) && !visited[x][y + 1] && image[x][y + 1]){
                  visited[x][y + 1] = 1;
                  floodFill(image,dimx,dimy,x,y + 1,visited);
            }
      }
}
// This is the function to be called
void floodFill(int **image, const int dimx, int const dimy,
               const int x, const int y){

      // We define this extra 2D-array to avoid visiting the same pixel
      // more than once in the above function
      int **visited = new int*[dimx];
      for (int i = 0; i < dimx; i++){
            visited[i] = new int[dimy];
            for (int j = 0; j < dimy; j++)
                  visited[i][j] = 0; // initially all pixels are unvisited
      }

      // mark the starting point as visited
      visited[x][y] = 1;
      floodFill(image,dimx,dimy,x,y,visited);

      for (int i = 0; i < dimx; i++)
            delete []visited[i];
      delete []visited;
}
```

Question 6: What is the time-complexity of the following code fragments in terms of big-O notation?

a)
```
for (i = 0; i <= n; i++){
     j = n;
     while (j >= i)
          j--;
}
```
Answer: O(n^2)

b)
```
for (i = 0; i <= n; i++){
     j = 0;
     while (j < 10000)
          j++;
}
```
Answer: O(n)

c)
```
void display(int n){
     for (int i = 0; i <= n; i++)
          for (int j = 0; j < n*n; j++)
               cout << i * j << endl;
}
void myfunction(int m){
     for (int i = 0; i <= m; i++)
          display(m);
}
```
Answer: O(m^4)

d)
```
void display(int n){
     for (int i = 0; i <= n; i++)
          for (int j = 0; j < n*n; j++)
               cout << i * j << endl;
}
void myfunction(int m){
     for (int i = 0; i <= m; i++)
          display(i);
}
```
Note that $\sum_{i=1}^{m} i^3 = \left(\sum_{i=1}^{m} i\right)^2 = \left(\frac{m(m+1)}{2}\right)^2$

Answer: O(m^4)

e)
```
int *arr = new int[n];
int k = 0;
for (int *i = arr; i < arr + n; i++)
     k++;
```
Answer: O(n)

f)
```
int f1(int n){
     if (n < 100)
          return 1;
     return n * f1(n-3);
}
```
Answer: O(n)

g)
```
int f2(int n){
     if (n <= 8)
          return 1;
     return n * f2(n/3);
}
```
Answer: O(log n)

h)
```
int f3(int n){
     if (n < 100)
          return 1;
     return n * f3(n-1) * f3(n-2) * f3(n-3);
}
```
Answer: O(3^n)