RECURSION WITH STRINGS

Q1. Reverse a string using recursion.

I/P : "abcdc"
O/P : "cdcba"

I/P : "babbar"
O/P : "rabbab"

I/P : "abccba"
O/P : "abccba"

Logic : Solve 1 case and identify base cases.

Our function will look something like this :

```
void reverse (string s , int i , int j) {

}
```

where i & j are the 2 indices to be swapped.

Thus, base case will be when i > j.

We solve the given sub-task by swapping elements at i and at j, and then incrementing & decrementing i and j respectively. Call the function again till we hit base case.

Code :

```cpp
#include<iostream>
using namespace std;

void reverse(string& str, int i, int j ) {

    //base case
    if(i>j)
        return ;

    swap(str[i], str[j]);
    i++;
    j--;

    //Recursive call
    reverse(str,i,j);

}

int main() {

    string name = "babbar";

    reverse(name, 0 , name.length()-1 );

    cout << name << endl;

    return 0;
}
```

Don't forget to pass the string by reference or else we will not be modifying the original string name

Usecase: Let name = "abcde"

① reverse ("abcde", 0, 4)

↳ abcde ⇒ ebcda
i ⟷ j

② reverse("ebcda", 1, 3)

↳ ebcda ⇒ edcba
(i ⇄ j)

③ reverse("edcba", 2, 2)

↳ edcba ⇒ edcba
(i = j)

④ reverse("edcba", 3, 1)

↳ Base Case hits.

We can also make our base case hit for i >= j

Homework: Improve the code by using up just one extra variable i instead of 2 (i & j).

Logic: Same as above just use i to get the element to be swapped. We can use the string's size to get the element.

First Element : i

Second Element : n - i - 1.

Code :

```cpp
#include <iostream>
using namespace std;

void reverse(string& s, int i) {
    if(i >= s.size()/2)
        return;
    // swap the i and n-i-1 indexed nodes
    swap(s[i], s[s.size() - i - 1]);
    // call for the next index
    reverse(s, i+1);
}

int main(void)
{
    string s;
    cin >> s;
    reverse(s, 0);
    cout << s << endl;
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
ork\Coding\Recursion\" ; if ($?) { g++ ReverseString.cp
abcdefg
gfedcba
```

Q2 Check Palindrome. (If the reversed string is same as the original string)

I/P : "abccba"

O/P : true

I/P : "abcbde"

O/P : false

Approach 1: Reverse string and check if its equal to the original string.

Time Complexity : O(n)
Space Complexity : O(n)

Approach 2: Take 2 pointers i & j pointing to the first and last element. Check if they are equal & then increment and decrement i & j respectively till i < j. If at some point str[i] != str[j], return false.

Code :

```
bool checkPalindrome(string str, int i, int  j) {

    //base case
    if(i>j)
        return true;

    if(str[i] != str[j])
        return false;
    else{
        //Recursive call
        return checkPalindrome(str, i+1,j-1);
    }
}
```

Homework: Write the recursive function using just one extra pointing variable.

Code :

```
#include <iostream>
using namespace std;

bool isPalindrome(string& s, int i) {
    if(i >= s.size()/2) return true;

    if(s[i] != s[s.size()-i-1])
        return false;

    return isPalindrome(s, i+1);
}

int main(void)
{
    string s;
    cin >> s;
    if(isPalindrome(s, 0)) {
        cout << "Palindrome\n";
    }
    else {
        cout << "Not Palindrome\n";
    }
    return 0;
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
abccba
Palindrome
```

(Similar to reverse string)

Note: The time complexity of accessing the size of a std::string in C++ is O(1).

q3. Find power(a,b) using recursion.

I/P : 2, 4

O/P : 16

Approach : We can represent a^b as follows:

$$a^b = \begin{cases} a \cdot a^{b-1} & \text{if } b \text{ is odd} \\ a^{b/2} \cdot a^{b/2} & \text{if } b \text{ is even} \end{cases}$$

We can use a recursive function with base case being $b = 0$ (or $b = 1$), because $a^0 = 1$ (or $a^1 = a$)

For a given call, if $b \% 2 == 0$ then return func(a, b/2) * func(a, b/2),

else return a * func(a, b-1) which equals

a * func(a, b/2) * func(a, b/2).

Confused ?

Example : power(2,5)

$2^5 = 2 \times 2^4$

$2^4 = 2^2 \times 2^2$

$2^2 = 2^1 \times 2^1$

$2^1 = 2$

When power(2,5) is called, we will perform 2 steps in one call to improve efficiency.

Instead of giving back 2 * power(2,4)

we will calculate pow(2,2) before-hand by using variable = power(2, 5/2);

Then return 2 * variable * variable.

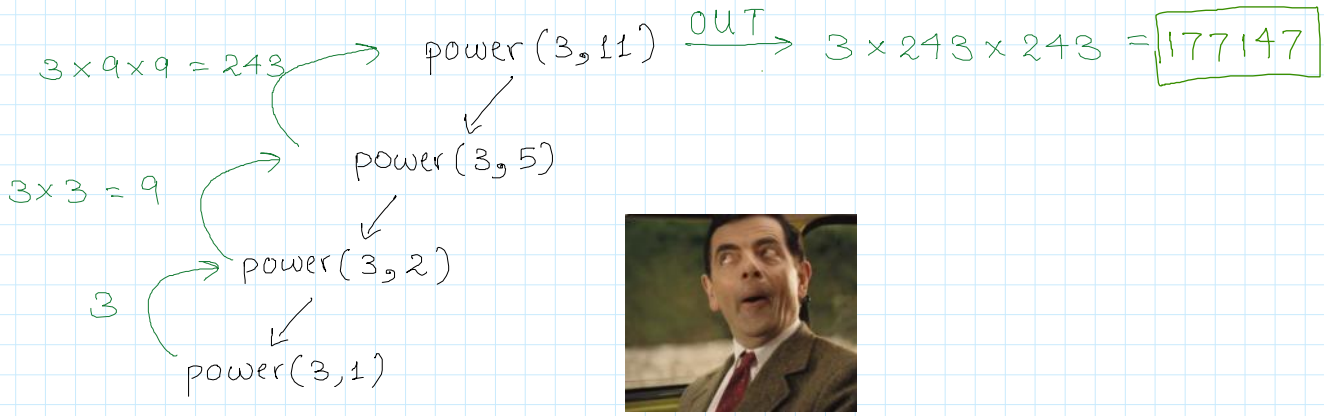
For even power,

return variable * variable.

Code :

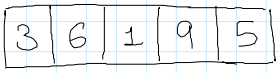
```
int power(int a, int b) {
    //base case
    if( b == 0 )
        return 1;

    if(b == 1)
        return a;

    //RECURSIVE CALL
    int ans = power(a, b/2);

    //if b is even
    if(b%2 == 0) {
        return ans * ans;
    }
    else {
        //if b is odd
        return a * ans * ans;
    }
}
```

Recursion Tree : power(3, 11)

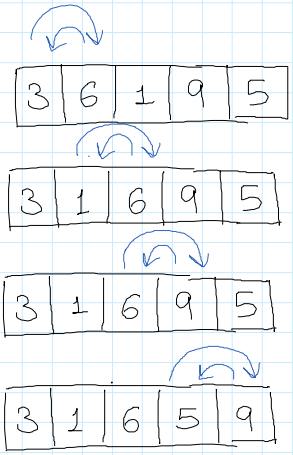


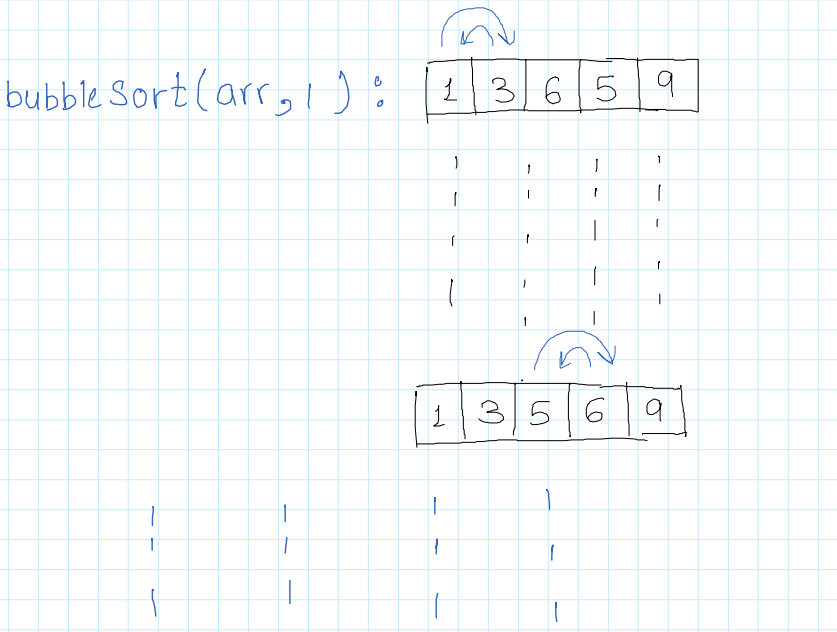
Q4 Bubble Sort using Recursion.

Approach : The recursive function will be called n times, each time it will correctly position the i^{th} largest element correctly. Base case hits when only 1 or no elements are left



bubbleSort(arr, 0) :





Code :

```
void sortArray(int *arr, int n) {

    //base case - already sorted
    if(n == 0 || n == 1) {
        return ;
    }

    //1 case sovle karlia - largest element ko end me rakh dega
    for(int i=0; i<n-1; i++) {
        if(arr[i] > arr[i+1]){
            swap(arr[i], arr[i+1]);
        }
    }

    //Recursive Call
    sortArray(arr, n-1);

}
```

Homework:

(1) Selection sort using recursion.

Code :

```
4   int minIndex(int a[], int i, int j)
5   {
6       if (i == j)
7           return i;
8       // minimum element's index from a[(i+1)....j]
9       int k = minIndex(a, i + 1, j);
10      // Comparing with current element and updating answer
11      return (a[i] < a[k])? i : k;
12  }
13
14  void recurSelectionSort(int a[], int n, int index = 0)
15  {
16      // Return when starting and size are same
17      if (index == n)
18          return;
19
20      // minimum element's index in a[index....(n-1)]
21      int k = minIndex(a, index, n-1);
22
23      swap(a[k], a[index]);
24
25      // Recursively calling selection sort function
26      recurSelectionSort(a, n, index + 1);
```

```
23      swap(a[k], a[index]);
24
25      // Recursively calling selection sort function
26      recurSelectionSort(a, n, index + 1);
27  }
28
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
5
3 1 4 5 2
1 2 3 4 5
```

Refer Here for detailed explaination of Homework Questions.