Common Mistakes in Recursion

In this document we discuss two common mistakes when writing recursive functions. We will provide two examples of incorrect binary search. These examples may look useless and contrived, since you already have a correct version of binary search and other well-known algorithms (Merge Sort, QuickSort) in the slides. But you may be required to write some variant of these algorithms, and knowing about the common mistakes may help you to avoid them when you write a variant of these well-known algorithms.

Mistake#1: Unreachable base case resulting in infinite recursion:

Consider the following **WRONG** code for recursive binary search:

```
int recursiveBinarySearch(int * A,int start,int end, int x)
{
if(start <=end) {
int mid = (start+end)/2;
if (A[mid] < x)
return recursiveBinarySearch(A,mid,end,x);
else if (A[mid] > x)
return recursiveBinarySearch(A,start,mid-1,x);
else if (A[mid] ==x)
return mid; // this is a base case
}
else return -1;// another base case of the recursion
}
```

This code is almost identical to the one given in the slides except that we replaced
`return recursiveBinarySearch(A,mid+1,end,x);`
by:
`return recursiveBinarySearch(A,mid,end,x);`

At first sight, the code still appears correct. It is true that if A[mid]<x , then the value x must have an index between mid and end, if it appears at all in the array.

Unfortunately this code isn't correct. Try it on the input A={0,1,2,3,4,5} and x= 5.
It will fail. Why ? Because the base case is unreachable: First the function is called with arguments start= 0 and end=5, then it will call itself recursively with start = 2 and end = 5, then it will call itself recursively with start = 3 and end = 5, then it will call itself with start = 4 and end =5 and then... it will call itself again with start = 4 and end =5, and the program will be stuck in an infinite recursion. Sometimes this failure will manifest itself very weirdly, for example by keeping the screen blank for a few seconds and then printing “Press any key to continue” and stopping the program. Unlike an infinite loop, infinite recursion cannot really go on very long, because each of the “workspaces” used by the recursive calls consumes memory, and very quickly the program will run out of memory. Such an error is called a stack overflow.
How do you do to avoid such errors ? The general strategy is very simple: You should make sure that

the recursive calls **ALWAYS** handle subproblems **strictly** smaller than the original problem, and you should make sure that the base case all problem sizes smaller than a certain threshold. In the above case the size of the right subarray was not always strictly smaller than the entire array, and that was the cause of the error.

Mistake #2: Making out-of-bounds recursive calls

This kind of error is common when you write recursive algorithms on arrays.
Very often, recursive algorithms on arrays are divide-and-conquer algorithms which work by dividing the original arrays in ever-smaller subarrays. For example, Binary Search, Quicksort, and Merge Sort work in this fashion. Subarrays are usually specified using their first index and their last index, and these two indexes are passed as arguments. **You have to make sure that you don't pass out-of-bound indexes as arguments in your recursive calls, and if you can't or don't want to avoid it, you should make sure your code can handle such invalid arguments.**

Example: Suppose we slightly modify the Binary Search function by adding the statement: if(A[start]==x) return start; at the very beginning of the function, and that we write the following program:

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

int B[] = {0,1,2,3,4,5,6,7,8,9,10};
int C[] = {11,0};

int recursiveBinarySearch(int * A,int start,int end, int x)
{

if(A[start]==x) return start; //Line L1
if(start <=end) {
int mid = (start+end)/2;
if (A[mid] < x)
return recursiveBinarySearch(A,mid+1,end,x);
else if (A[mid] > x)
return recursiveBinarySearch(A,start,mid-1,x);
else
return mid; // this is a base case
}
else return -1;// another base case of the recursion
}

int main(){

        int a = recursiveBinarySearch(B,0,10,11);
        cout << a << endl << flush;
        return 0;
}
```

At first sight, the Binary Search function still appears to be correct, although the first line of the function is not necessary. But if you run the above program, the output will be wrong (It will return 11, when the last valid index in array B is only 10, and the function should have returned -1). Why ?

Because at some point, the start index will be outside the array. This fact is often overlooked but even if the line we added (L1) is removed, it is still true that the start variable will be greater than 10 at some time. To verify this, add a cout statement at the beginning of the recursive function to check the value of the arguments. And if B={10,11,12} and x = -1, the recursiveBinarySearch function will at some point be recursively called with a negative end index. However if Line L1 is omitted, these out-of-bounds values will not cause any error because they will not be used to index any element in the array, as the condition if(start<=end) will filter out these cases: The first time either start or end will become an invalid index, the condition (start<=end) will become false (This may not be obvious at first but will become clear with a bit of thinking.) This is so because the function was written in a way that handles this problem.
OK, now assume that you have written your own recursive function, and that it does not return the correct answer, and you suspect that the code contains an out-of-bounds error. How can you find this error ? One good way would be to add a cout statement at the beginning of the recursive function, to display its arguments, especially the ones corresponding to subarray bounds, and run the program to see if some of these bounds look wrong. for example :

```
int recursiveBinarySearch(int * A,int start,int end, int x)
{
cout << " start = " << start << " end =  "  <<  end <<  endl << flush;
if(A[start]==x) return start; //Line L1
if(start <=end) {
int mid = (start+end)/2;
if (A[mid] < x)
return recursiveBinarySearch(A,mid+1,end,x);
else if (A[mid] > x)
return recursiveBinarySearch(A,start,mid-1,x);
else
return mid; // this is a base case
}
else return -1;// another base case of the recursion
}
```

Please note that the cout statement ends with << flush . This is necessary when you use cout for debugging purposes. This tells the computer to display the result as soon as the cout statement is executed. If flush is omitted, the computer will sometimes store the output in a buffer, and will wait until the function stops executing to display the contents of the buffer; but if the functions fails in a runtime error, the contents of the buffer will not be displayed at all.