XJTLU | Computing

CPT108: Data Structures and Algorithms

Semester 2, 2023-24

Tutorial 1

Recursive Problem Solving

1. Suppose you are climbing the stairs. It takes you n steps to get to the roof ($n \geq 1$). You can climb 1 or 2 steps at a time. How many different ways can you get to the top?

   (a) Consider the base case. How many ways can you get to the first step? How about the second step?

   (b) Consider the state transition. Suppose there are $f(n-2)$ ways to get to the $(n-2)^{th}$ step, and $f(n-1)$ ways to get to the $(n-1)^{th}$ step, how many ways can we get to the n^{th} floor? Compute $f(n)$ with $f(n-2)$ and $f(n-1)$.

   (c) Complete the following code with above conclusions.

```
public int climbStairs(int n) {
    if (n==1) return ______;
    if (n==2) return ______;
    return ______;
}
```

   (d) Give an integer array cost, where `cost[i]` is the amount you pay to climb up from the i^{th} step of the staircase. Once you pay this fee, you can choose to climb one or two steps up. You can choose to climb the stairs starting from step 0 or 1. Complete the following code to calculate and return the minimum cost to reach the top of the stairs.

```
public int minCostClimbingStairs(int[] cost) {
    return this.minCostClimbingStairsRecursive(cost, cost.length);
}


public int minCostClimbingStairsRecursive(int[] cost, int n){
  // your implementation





}
```

Code Tracing

2. The following method implements a bubble sort algorithm.

```java
public void bubbleSort(int[] array){
    for(int i=0; i<array.length; i++){
        for(int j=0; j<array.length-1; j++){
            if(array[j]>array[j+1]){
                int tmp = array[j+1];
                array[j+1] = array[j];
                array[j] = tmp;
            }
        }
    }
}
```

(a) Given the input array as $[10, 5, 4, 3]$, trace the array after each iteration.

i	j	array
0	0	5, 10, 4, 3
0	1	5, 4, 10, 3
0	2	
1	0	
1	1	
1	2	
2	0	
2	1	
2	2	
3	0	
3	1	
3	2	

(b) In terms of the above bubble sort algorithm, given an array with length as n, for each iteration of the outer loop, how many times does the inner loop iterate? Observing the table, is it necessary for the inner loop to iterate so many times? If not, try to find the minimum number of iterations of the inner loop, and ensure that any array can be sorted successfully under this number of iterations (Suppose the index of outer loop is i).

3. The following code is intended to find the position of the number equals to the target within an ascending array.

```java
public int binarySearch(int[] nums, int target) {
    int left = 0, right = nums.length - 1;
    while (left <= right) {
        int mid = (right - left) / 2 + left;
        int num = nums[mid];
        if (num == target) {
            return mid;
        } else if (num > target) {
            right = mid;
        } else {
            left = mid;
        }
    }
    return -1;
}
```

Trace the code with `nums` as $[2, 4, 5, 7, 10, 13]$ and `target` as 13, what errors do you find, and how you fix the code?

4. The following method 'search' is intended to find the position of the first number that larger than the target within an ascending array (e.g., given an array $[0, 2, 5, 7]$ and target as 3, the method should output 2 since 5 is the first element that larger than 3).

```java
public int search(int[] nums, int target) {
    return firstLarger(nums, 0, nums.length-1, target);
}

public int firstLarger(int[] nums, int left, int right, int
   target){
    if (nums.length == 1){
        if (nums[0] > target)
            return 0;
        else
            return -1;
    }
    int mid = (right-left)/2 + left;
    if (nums[mid] <= target){
        return firstLarger(nums, mid+1, right, target);
    } else {
        if(nums[mid-1] <= target)
            return mid;
        return firstLarger(nums, left, mid-1, target);
    }
}
```

(a) Trace the code with:
   (1) `nums` as $[-1, 0, 3, 5, 9, 12]$ and `target` as 8;
   (2) `nums` as $[1, 3, 4]$ and `target` as 0.
   What error do you find, and how you fix the code?
(b) What is the time complexity of this method?

Solutions

1. (a) There are one way to get to the first step (start from step 0 and climb one step) and two ways to get to the second step (1. start from step 0 and climb two step; or 2. start from step 0, climb one step and another step).

   (b) $f(n) = f(n-2) + f(n-1)$

   (c)
```
public int climbStairs(int n) {
    if (n==1) return 1;
    if (n==2) return 2;
    return this.climbStairs(n-2) + this.climbStairs(n-1);
}
```

   (d) First, considering the base case. Suppose there is only one or two steps, we can start from either of them without any expense.

   Now considering the state transition, suppose we have figured out the minimum cost to step $n-2$ and step $(n-1)$ as $f(n-2)$ and $f(n-1)$. There are two ways to get to step n. One is starting from step $n-2$, pay the fee `cost[n-2]` and climb 2 steps, the other way is starting from step $n-1$, pay the fee `cost[n-1]` and climb 1 steps. The minimum cost to step n is right the smaller one between $f(n-2)+$ `cost[n-2]` and $f(n-1)+$ `cost[n-1]`. Hence, we have the following code:

```
public int minCostClimbingStairs(int[] cost) {
    return this.minCostClimbingStairsRecursive(cost, cost.length);
}

public int minCostClimbingStairsRecursive(int[] cost, int n){
    if (n == 0 || n == 1) return 0;
    return Math.min(this.minCostClimbingStairsRecursive(cost, n-1) +
        cost[n-1],
            this.minCostClimbingStairsRecursive(cost, n-2) + cost[n-2])
                ;
}
```

2. (a)

i	j	array
0	0	5, 10, 4, 3
0	1	5, 4, 10, 3
0	2	5, 4, 3, 10
1	0	4, 5, 3, 10
1	1	4, 3, 5, 10
1	2	4, 3, 5, 10
2	0	3, 4, 5, 10
2	1	3, 4, 5, 10
2	2	3, 4, 5, 10
3	0	3, 4, 5, 10
3	1	3, 4, 5, 10
3	2	3, 4, 5, 10

   (b) In the above algorithm, the inner loop iterates for $n-2$ times for each iteration of the outer loop. It is unnecessary to iterate for so many times. The minimum iteration time of the inner loop should be $n-i-2$. Since after the outer loop iterate for i times, the last i items of the array are well-sorted and need not be revisited.

3.

left	right	mid	num	region
0	5	2	5	2,4,5,7,10,13
2	5	3	7	5,7,10,13
3	5	4	10	7,10,13
4	5	4	10	7,10,13

The program would get stuck when `mid` equals to 4. To fix it, we can modify line 9 and line 11 as '`right` = `mid` + 1' and '`left` = `mid` - 1', respectively.

4. (a) (1)

left	right	mid	nums[mid]	nums[mid-1]	region	return
0	5	2	3	0	-1, 0, 3, 5, 9, 12	
3	5	4	9	5	5, 9, 12	4

(2)

left	right	mid	nums[mid]	nums[mid-1]	region	return
0	2	1	3	1	1, 3, 4	
0	1	0	1	null	1	error

The code has the risk of `IndexOutOfBoundsException` caused by line 16. To fix it, we can modify line 16 as:

```
            if(mid < 1 || nums[mid-1] <= target)
```

(b) The time complexity is $O(\log n)$ at the worst case. For a recursive algorithm, the core to analyze its time complexity is to counts the number of method call. In this case, in the worst case, suppose the length of the array is n and we call the `firstLarger` method for x times, naturally we have $2^x = n$, i.e., $x = \log_2 n$.