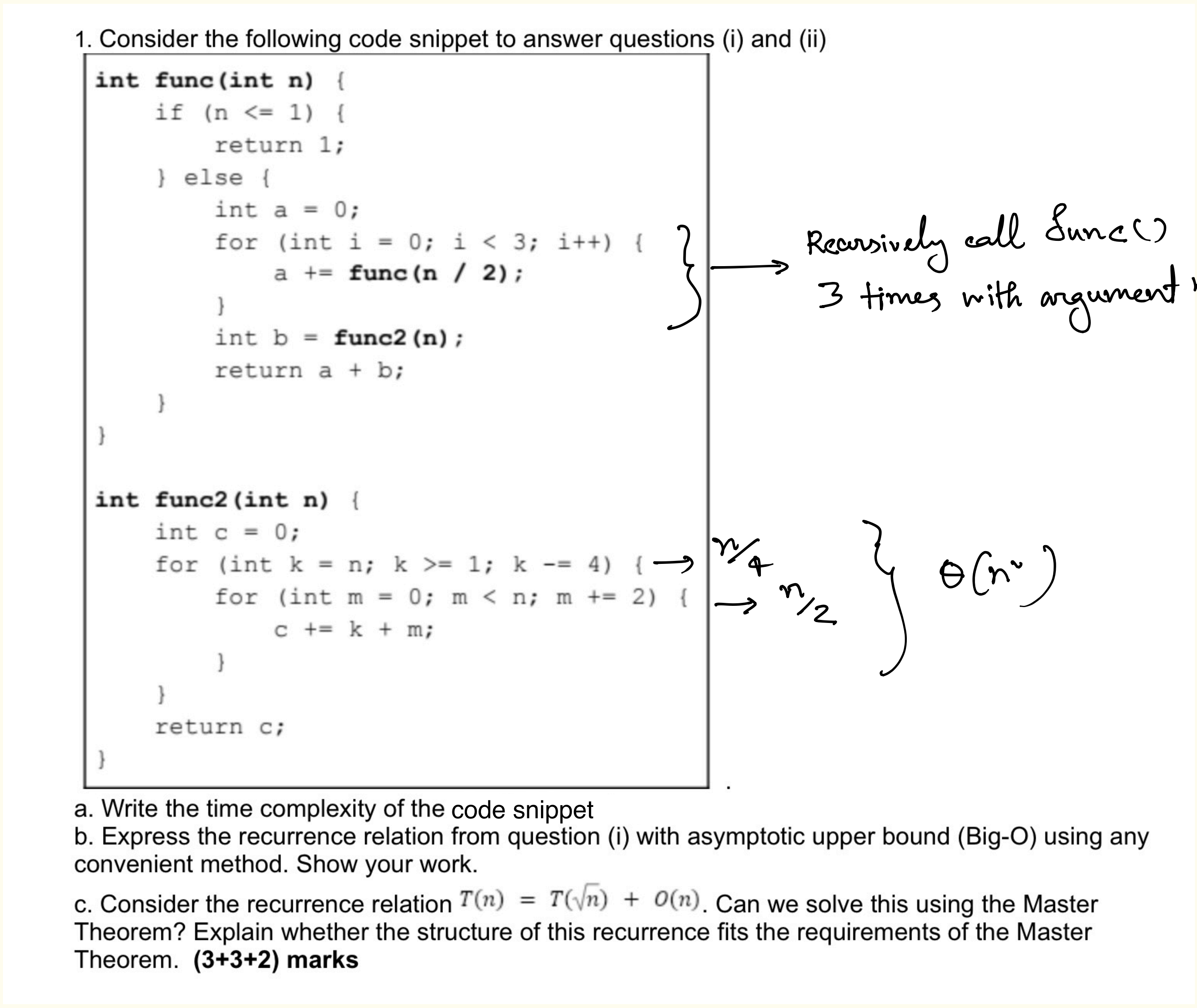


1. Consider the following code snippet to answer questions (i) and (ii)

```
int func(int n) {
    if (n <= 1) {
        return 1;
    } else {
        int a = 0;
        for (int i = 0; i < 3; i++) {
            a += func(n / 2);
        }
        int b = func2(n);
        return a + b;
    }
}

int func2(int n) {
    int c = 0;
    for (int k = n; k >= 1; k -= 4) {
        for (int m = 0; m < n; m += 2) {
            c += k + m;
        }
    }
    return c;
}
```

Recursively call func() 3 times with argument $n/2$

$n/4$

$n/2$

$\Theta(n^2)$

a. Write the time complexity of the code snippet
b. Express the recurrence relation from question (i) with asymptotic upper bound (Big-O) using any convenient method. Show your work.
c. Consider the recurrence relation $T(n) = T(\sqrt{n}) + O(n)$. Can we solve this using the Master Theorem? Explain whether the structure of this recurrence fits the requirements of the Master Theorem. **(3+3+2) marks**

a, b) Recurrence Relation,

$$T(n) = 3T(n/2) + \Theta(n^2)$$

from master theorem,

$$T(n) = aT(n/b) + \Theta(n^k \log^p n)$$

$a=3$, $b=2$, $k=2$, $p=0$

$b^k = 4$. So, $a < b^k$ and $p=0$

$$T(n) = \Theta(n^k \log^p n)$$

$$T(n) = \Theta(n^2)$$

c) $T(n) = \underline{T(\sqrt{n})} + O(n)$ $\sqrt{n}$ Doesn't match with the format n/b

3. a) Each robot can form connections with some of the other robots. That means each robot can form at most (n-1) connections. Which algorithm will be the fastest to sort the robots based on the number of connections they formed? Write the name and time complexity of the algorithm. **2 marks**

Counting Sort

Time complexity $\rightarrow O(n)$

b) Suppose, in the factory, there exists a system that can find the tallest robot in O(1) time. That means you are given a function, find_tallest(A), which returns the tallest robot from a list called A, in constant time.
Considering this information, design an algorithm to sort the robots in descending order of their heights. The time complexity of your algorithm must not exceed O(n).
i. Present your algorithm using pseudocode/ executable code/ step-by-step logical instructions.
ii. Write the time complexity of your algorithm. **(2+1) marks**

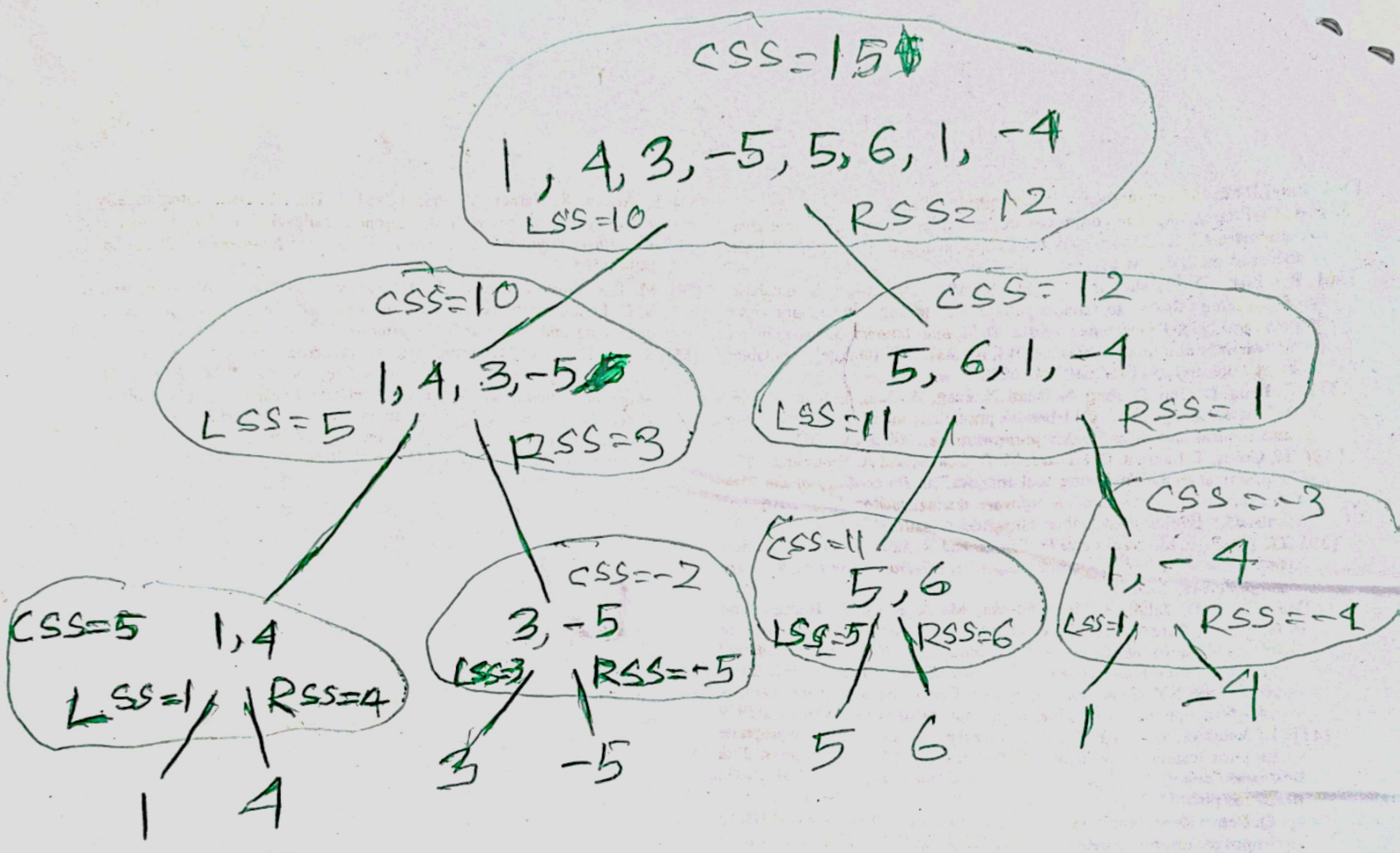
b) i)
```
modified_selection_sort( A )
        for i=1 to n :
            idx, val = find_tallest (A[i ....n]);
            swap( A[i], A[idx]);
```

ii) $O(n)$

2. A football team’s analytics department hired you and gave you a task of reviewing match performance data from the past season. Each match's net goal difference (goals scored minus goals conceded) is recorded in a list. Now you were asked by the department to identify the team’s best consecutive performance streak, where the net goal difference was maximized over a series of matches.

net_goal_difference: [1, 4, 3, -5 , 5, 6, 1, -4]

To prove your skills, you used an approach where the problem is divided into subproblems to find the solution. Now tell us more about it by answering the following question.
For the given input above, determine the number of times the Cross-Sum exceeds both the Left-Sum and the Right-Sum. Exclude base cases (subarrays of size 1) from consideration. Show your work properly.
Cross-Sum: sum of elements calculated by combining sum from mid to left and mid to right of the current subarray.
Left-Sum: maximum sum in the left half of the subarray.
Right-Sum: maximum sum in the right half of the subarray. **7 marks**


CSS=15
1, 4, 3, -5, 5, 6, 1, -4
LSS=10
RSS=12
CSS=10
1, 4, 3, -5
LSS=5
RSS=3
CSS=12
5, 6, 1, -4
LSS=11
RSS=1
CSS=5
1, 4
LSS=1
RSS=4
CSS=-2
3, -5
LSS=3
RSS=-5
CSS=11
5, 6
LSS=5
RSS=6
CSS=-3
1, -4
LSS=1
RSS=-4
1
4
3
-5
5
6
1
-4