CS 218, Spring 2017 Homework 1
Abhishek Kumar SRIVASTAVA April 12, 2017
Student ID: 861307778

I certify that this submission represents my own original work

---

Solution 1. ***Step 1:*** Initially $i = 1$. In the **Loop 1**(while $i \leq n$ do), the value of i increases by the power of 2 till it reaches n. Since it is given that we can assume n is a power of 2, then let m be a number such that $2^m = n$.

Step 2: For each iteration of **Loop 1**, in **Loop 2**(for $j = i$ to $2i - 1$) the value of j goes from i to $2i - 1$, so this loop runs $2i - 1 - i + 1 = i$ times.

Step 3: Therefore, cost of **Loop 2** is given by the sum of $O(1)$ i.e Constant time print operations, since no other operation is inside the loop. Therefore,

$$C_2 = \sum_{j=i}^{2i-1} O(1)$$

Since it runs i number of times, it is sum of constant values i number of times.

$$\begin{aligned} &= O(1) + O(1) + ... + O(1) \\ &= i \end{aligned}$$

Step 4: So the cost of **Loop 1** is given by the sum of **Loop 2** cost(C_2) for value i takes. Therefore,

$$\begin{aligned} C_1 = \sum_{i=1}^{n} C_2 &= \sum_{i=1}^{n} i \\ &= 1 + 2 + 4 + ...n \end{aligned}$$

Since i increase by the power of 2 and we have assumed that $n = 2^m$.

$$\begin{aligned} &= 1 + 2 + 4 + ...2^m \\ &= \frac{2^{m+1} - 1}{2 - 1} \\ &= 2^{m+1} - 1 = 2.2^m - 1 && \text{(Substituting } n = 2^m\text{)} \\ &= 2.n - 1 \leq 2.n \end{aligned}$$

Step 6: Therefore, the total cost is cost of **Loop 1**(C_1) which runs in less than $2n$ times for a given n. Let, $T(n) = n$. To prove the tight bounds, there must exist c_1 and c_2 such that $c_1 \times g(n) \leq T(n) \leq c_2 \times g(n)$.

$$n \leq n < 2n$$

Therefore, $c_1 = 1$, $c_2 = 2$ and $g(n) = n$. Hence, $T(n) \in \Theta(n)$.

Solution 2. Since the wall is stretched infinitely in the both direction i have assumed the starting location as 0. From the starting point we will go 2^i steps first in the right side, come back to starting location and then we will go 2^i steps in the left side and return back to starting location and increase the value of i by 1 where $i = 0, 1...m$ which is nothing but doubling the steps to be taken in the next iteration for either side, we will continue until the door is found. One more assumption i am taking is while going from starting location to the 2^ith location we will check for the doors at each location but skipping the locations which are already checked that is doors between 0 to 2^{i-1}. Psuedo Code to solve this problem is given on page 3.

Part 1.

The worst case will be when the door is on the left side and let us assume its value $n = 2^m + d$, where $1 \leq d \leq 2^m$ Since we are traversing in both direction the total cost would be:

$$T(n) = 2 \cdot (2 \cdot (1 + 2 + 4...2^m)) + 2 \cdot 2^{m+1} + 2^m + d$$

Inner bracket cost is because we are going to 2^ith location and coming back and it is multiplied by 2 because we are doing it for both directions i.e Right and Left. $2 \cdot 2^{m+1}$ because we are going till 2^{m+1} times in the right direction and then coming back to the starting location. $2^m + d$ is added because we are going till 2^m location in the left direction and next d steps to find the door location. Therefore,

$$T(n) = 4 \cdot (1 + 2 + 4...2^m) + 2 \cdot 2^{m+1} + 2^m + d$$

Which is equivalent to the following equation.

$$T(n) = 4 \cdot (2^{m+1} - 1) + 4 \cdot 2^m + 2^m + d$$

After simplifying in 2^m terms.

$$T(n) = 8 \cdot 2^m - 4 + 4 \cdot 2^m + 2^m + d$$

$$T(n) = 13 \cdot 2^m + d - 4$$

And since $n = 2^m + d$,$2^m = n - d$. By replacing it we can get.

$$T(n) = 13 \cdot (n - d) + d - 4$$

$$T(n) = 13 \cdot n - 12 \cdot d - 4$$

$$T(n) \leq 13 \cdot n$$

$\therefore T(n) = O(n)$

Part 2.

Since we already got $T(n)$ expression from the Part 1 which is

$$T(n) = 13 \cdot n - 12 \cdot d - 4$$

The worst case will be when $d = 1$.Which will the equation as,

$$T(n) = 13 \cdot n - 16$$

We can see the the equation that coefficient of n is 13. which is the constant multiple in the worst case.

```
procedure FIND_DOOR_LOCATION(A)
    if door at A[0] = true then                                   ▷ Base case
        door_location ← 0
        return door_location
    end if
    pos ← 0                                                       ▷ starting position
    is_door_found ← false                                         ▷ True if door found
    door_location ← 0
    steps_to_take ← 1                                             ▷ How much step to take
    while is_door_found ≠ true do
        Traverse in Right Direction
        while pos ≤ steps_to_take/2 do                            ▷ Skipping previously checked doors
            pos + +
        end while
        while pos ≤ steps_to_take do                              ▷ Check Right Direction
            pos + +
            if door at A[pos] = true then                         ▷ Check Each location
                door_location ← pos
                is_door_found ← true
                break
            end if
        end while
        if is_door_found ≠ true then                              ▷ If door not found in Right Side
            while pos ≠ 0 do                                      ▷ Traversing back to starting position
                pos − −
            end while
            Traverse Left Direction
            while pos ≤ steps_to_take/2 do                        ▷ Skipping previously checked doors
                pos + +
            end while
            while pos ≤ steps_to_take do                          ▷ Check Left Direction
                pos + +
                if door at A[pos] = true then                     ▷ Check Each location
                    door_location ← pos
                    is_door_found ← true
                    break
                end if
            end while
        end if
        if is_door_found ≠ true then                              ▷ If door also not found Left Side
            while pos ≠ 0 do                                      ▷ Traversing back to starting position
                pos − −
            end while
            steps_to_take ← 2 ∗ steps_to_take                     ▷ Increasing steps to be taken by 2
        end if
    end while
    return door_location
end procedure
```

Solution 3. 1. Iterative substitutions
Given that

$$
\begin{aligned}
T(n) &= \begin{cases} 1 & n = 1 \\ 4 \cdot T(\frac{n}{2}) + 3 & n > 1 \end{cases} \\
&= 4 \cdot (4 \cdot T(\frac{n}{4}) + 3) + 3 \\
&= 16 \cdot T(\frac{n}{4}) + 4 \cdot 3 + 3 \\
&= 16 \cdot (4 \cdot T(\frac{n}{8}) + 3) + 4 \cdot 3 + 3 \\
&= 64 \cdot T(\frac{n}{8}) + 16 \cdot 3 + 4 \cdot 3 + 3 \\
&. \\
&. \\
&. \\
&= 4^i . T(\frac{n}{2^i}) + 4^{i-1} \cdot 3 + 4^{i-2} \cdot 3 + ... + 4^0 \cdot 3 \\
&= 4^i \cdot T(\frac{n}{2^i}) + 3 \cdot (4^{i-1} + 4^{i-2} + ... + 4^0)
\end{aligned}
$$

Geometric series for ratio r = 4

$$
\begin{aligned}
&= 4^i \cdot T(\frac{n}{2^i}) + 3 \cdot (\frac{4^i - 1}{4 - 1}) \\
&= 4^i \cdot T(\frac{n}{2^i}) + 4^i - 1
\end{aligned}
$$

$i = \log_2(n)$ will give us the base case,

$$
\begin{aligned}
&= 4^{\log_2(n)} \cdot T(1) + 4^{\log_2(n)} - 1 && \text{(since } i = \log_2(n)\text{)} \\
&= 2 \cdot 4^{\log_2(n)} - 1 \\
&= 2 \cdot 2^{\log_2(n^2)} - 1 && \text{(log manipulation)} \\
\therefore T(n) &= 2 \cdot n^2 - 1
\end{aligned}
$$

2. **Proof by induction**
Given that

$$
T(n) = \begin{cases} 1 & n = 1 \\ 4 \cdot T(\frac{n}{2}) + 3 & n > 1 \end{cases}
$$

From Iterative Substitution: $T(n) = 2 \cdot n^2 - 1$

Proof: Base case: $n = 2$
Calculate using given recurrence relation.

$$
\begin{aligned}
T(2) &= 4 \cdot T(\frac{2}{2}) + 3 \\
T(2) &= 4 \cdot 1 + 3 = 7 && \text{Since, T(1) = 1}
\end{aligned}
$$

Calculate using derived asymptotic solution.

$$
\begin{aligned}
T(2) &= 2 \cdot n^2 - 1 \\
T(2) &= 2 \cdot 2^2 - 1 = 7 && \text{Since, n = 2}
\end{aligned}
$$

Since both result are the same hence derived solution holds.

Induction hypothesis: T($\frac{n}{2}$) $= 2 \cdot (\frac{n}{2})^2$ - 1 $= \frac{n^2}{2} - 1$

Induction step:

$$\begin{aligned}
T(n) &= 4 \cdot T(\frac{n}{2}) + 3 \\
&= 4 \cdot (\frac{n^2}{2} - 1) + 3 && \text{(Substituting hypothesis)} \\
&= 4 \cdot (\frac{n^2}{2}) - 4 + 3 \\
&= 2 \cdot n^2 - 1
\end{aligned}$$

Which is same as our hypothesis.

$\therefore T(n) = 2 \cdot n^2 - 1.$