Lecture 3_Algorithms

Search

1. **What is search/searching?**
   Process of finding a number, character, string, or other item is called **searching** .
2. **Linear Search**
   1. Search one by one, from left to right or from right to left
3. **Binary Search**
   2. Requirement: The data in the array need to be arranged in a sequence(decreasing or increasing)
   3. Compare the target with the content of the middle of the array

Running time

1. *Running time* involves an analysis using *big O* notation, which shows how much time it takes an algorithm to solve a problem.
2. Computer scientists discuss efficiency in terms of *the order of* various running times.
3. Some common running times
   - $O(n^2)$
   - $O(nlog(n))$
   - $O(n)$
   - $O(log(n))$
   - O(1)
4. Linear search was of order $O(n)$ because it could take *n* steps in the worst-case to run. Binary search was of order $O(log(n))$ because it would take fewer and fewer steps to run, even in the worst-case.
5. We like to focus on the best case and the worst case, where big *O* denotes the worst case and Ω denotes the best case. The Θ symbol is used to denote where the upper bound and lower bound are the same: Where the best-case and the worst-case running times are the same.

Struct

1. C allows us to create our own data types via a **struct**.
2. An example: Our own datatype is called a `person` that has a string called `name` and another string called `number` .

```
typedef struct
{
    string name;
    string number;
} person;
```

3. To access the string inside the new datatype, we use dot `.`

```
person people[3];
people[0].name = "Yuliia";
people[0].number = "+1-617-495-1000";
```

Sorting

1. **What is sorting?**
   Sorting is the act of taking an unsorted list of values and transforming this list into a sorted one.
2. **Selection Sorting**
   1. Pseudocode
      `For i from 0 to n-1 Find smallest number between numbers[i] and numbers[n-1] Swap smallest number with numbers[i]`
   2. Running time

      $$t = \frac{n(n-1)}{2}$$

      or simply

      $$O(n^2)$$

      The running time is the same in the worst case and the best case, so it is

      $$\Omega(n^2)$$

3. **Bubble Sort**
   1. Pseudocode

```
Repeat n-1 times
For i from 0 to n-2
    If numbers[i] and numbers[i+1] out of order
        Swap them
```

```
If no swaps
    Quit
```

   2. Running time

      In the worst-case, or upper-bound, bubble sort is in the order of $O(n^2)$. In the best-case, or lower-bound, bubble sort is in the order of $\Omega(n)$.

4. **Merge Sort**
   1. Pseudocode

      ```
      If only one number
          Quit
      Else
          Sort left half of number
          Sort right half of number
          Merge sorted halves
      ```

   2. Running time

      Merge sort is a very efficient sort algorithm with a worst-case of $O(nlogn)$. The best-case is still $\Omega(nlogn)$ because the algorithm still must visit each place in the list. Therefore, merge sort is also $\Theta(nlogn)$ since the best-case and worst-case are the same.

Recursion

1. *Recursion* is a concept within programming where a function calls itself.
2. Example

```c
// Draws a pyramid using recursion

#include <cs50.h>
#include <stdio.h>

void draw(int n);

int main(void)
{
    // Get height of pyramid
    int height = get_int("Height: ");

    // Draw pyramid
    draw(height);
```

```
}

void draw(int n)
{
    // If nothing to draw
    if (n <= 0)
    {
        return;
    }

    // Draw pyramid of height n - 1
    draw(n - 1);

    // Draw one more row of width n
    for (int i = 0; i < n; i++)
    {
        printf("#");
    }
    printf("\n");
}
```

Takeaways

1. ***Segmentation fault:*** A part of memory was touched by your program that it should not have access to