Notes for ECE 36800 - Data Structures

Zeke Ulrich

January 23, 2025

Contents

Course Description

Provides insight into the use of data structures. Topics include stacks, queues and lists, trees, graphs, sorting, searching, and hashing.

Introduction

Algorithms

An algorithm is simply a method to solve a class of problems. There are algorithms to make toast, to solve a Rubik's cube, and to plot the optimal path for a mailmain. The two most important qualities of any algorithm are effectiveness and secondly efficiency. Our solution must work, preferably in as little time as possible. A working algorithm is no good if the heat death of the universe occurs before it finds a solution. We describe the cost of an algorithm using Big O notation. As an example, consider listing 1 If each operation has cost C_i, then

```
int total = 0; // C_1
for (int i = 0; i++; i <= n){ // C_2
    total = total + i; // C_3
}
return total; // C_4
```

Figure 1: Sum of first n numbers

the total cost of the program is

$$C_1 + C_2(n+1) + C_3 n + C_4 = n(C_2 + C_3) + (C_1 + C_4). \qquad (1)$$

Big O notation considers only the largest power of n, so the Big O complexity for this algorithm would be $O(n)$.

Data Structures

A data structure is an organization of information for ease of manipulation. For example, a dictionary, a checkout line, and an org chart.

Sorting

Bubble Sort

Bubble Sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.

Time Complexity:

- Best Case: $O(n)$ (when the array is already sorted)

- Average Case: $O(n^2)$

- Worst Case: $O(n^2)$

Listing 1: Bubble Sort

```
void bubble_sort(int arr[], int n) {
    for (int i = 0; i < n-1; i++) {
        int swapped = 0;
        for (int j = 0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
                swapped = 1;
            }
        }
        if (swapped == 0)
            break;
    }
}
```

Insertion Sort

Insertion Sort builds the sorted array one item at a time. It picks an element and places it into its correct position in the sorted part of the array.

Time Complexity:

- Best Case: $O(n)$ (when the array is already sorted)

- Average Case: $O(n^2)$

- Worst Case: $O(n^2)$

Listing 2: Insertion Sort

```
void insertion_sort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Selection Sort

Selection Sort divides the array into a sorted and an unsorted region. It repeatedly selects the smallest (or largest) element from the unsorted region and moves it to the sorted region.

Time Complexity:

- Best Case: $O(n^2)$

- Average Case: $O(n^2)$

- Worst Case: $O(n^2)$

Listing 3: Selection Sort

```
void selection_sort(int arr[], int n) {
    for (int i = 0; i < n-1; i++) {
        int min_idx = i;
        for (int j = i+1; j < n; j++) {
            if (arr[j] < arr[min_idx])
                min_idx = j;
        }
        int temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}
```

Shell Sort

Shell Sort is an optimization over Insertion Sort. It first sorts elements that are far apart and then progressively reduces the gap between elements to be sorted.

Time Complexity:

- Best Case: $O(n \log n)$

- Average Case: depends on the gap sequence, commonly $O(n^{3/2})$ or $O(n \log^2 n)$

- Worst Case: $O(n^2)$

Listing 4: Shell Sort

```c
void shell_sort(int arr[], int n) {
    for (int gap = n/2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {
                arr[j] = arr[j - gap];
            }
            arr[j] = temp;
        }
    }
}
```

A sorting algorithm is stable if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted. Insertion sort, merge sort, and bubble sort are all stable sorting algorithms.