Algorithms Lab Assignment 2

Pranav Gade

February 17, 2022

Batch: CS&AI
Roll no.: LCI2020010

1 Quick sort

Analysis of Quick sort complexity.

1.1 Code

```c
#include <stdlib.h>
#include <stdio.h>

void sort(int* arr, int size);

int main(int argc, char** argv) {
    int size = argc - 1;
    int* arr = malloc(sizeof(int) * size);
    for (int i = 1; i <= size; ++i) {
        arr[i-1] = atoi(argv[i]);
    }

    sort(arr, size);

    for (int i = 0; i < size; ++i) {
        printf("%d ", arr[i]);
    }

    free(arr);
    return 0;
}

void quick_sort(int* arr, int size) {
    if (size < 2) return;

    int pivot = arr[size];
    int i = -1;
    for (int j = 0; j < size; j++) {
        if (arr[j] <= pivot) {
            i++;
            int t = arr[i];
            arr[i] = arr[j];
```

```
            arr[j] = t;
        }
    }

    i++;
    int t = arr[i];
    arr[i] = arr[size];
    arr[size] = t;

    quick_sort(arr, i - 1);
    quick_sort(&arr[i + 1], size - (i + 1));
}

void sort(int* arr, int size) {
    quick_sort(arr, size-1);
}
```

1.2 Output

```
[p@claret cmake-build-debug]$ ./quicksort 4 6 2 8 3 0 1 2
0 1 2 2 3 4 6 8
```

Figure 1: Quick sort output

1.3 Graph

Figure 2: Quick sort runtime v/s input size plot

2 Merge sort

Analysis of Merge sort complexity.

2.1 Code

```c
#include <stdlib.h>
#include <stdio.h>

void sort(int* arr, int size);

int main(int argc, char** argv) {
    int size = argc - 1;
    int* arr = malloc(sizeof(int) * size);
    for (int i = 1; i <= size; ++i) {
        arr[i-1] = atoi(argv[i]);
    }

    sort(arr, size);

    for (int i = 0; i < size; ++i) {
        printf("%d ", arr[i]);
    }

    free(arr);
    return 0;
}


void merge_sort(int* arr, int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;

        merge_sort(arr, l, m);
        merge_sort(arr, m + 1, r);

        int n1 = m - l + 1;
        int n2 = r - m;
        int l1[n1], m1[n2];

        for (int i = 0; i < n1; i++) l1[i] = arr[l + i];
        for (int j = 0; j < n2; j++) m1[j] = arr[m + 1 + j];

        int i, j, k;
        i = 0;
        j = 0;
        k = l;

        while (i < n1 && j < n2) {
            if (l1[i] <= m1[j]) {
                arr[k] = l1[i];
                i++;
            } else {
                arr[k] = m1[j];
                j++;
            }
            k++;
        }

        while (i < n1) {
            arr[k] = l1[i];
```

```
            i++;
            k++;
        }

        while (j < n2) {
            arr[k] = m1[j];
            j++;
            k++;
        }
    }
}
void sort(int* arr, int size) {
    merge_sort(arr, 0, size-1);
}
```

2.2 Output

```
[p@claret cmake-build-debug]$ ./mergesort 4 6 2 8 3 0 1 2
0 1 2 2 3 4 6 8
```

Figure 3: Merge sort output

2.3 Graph

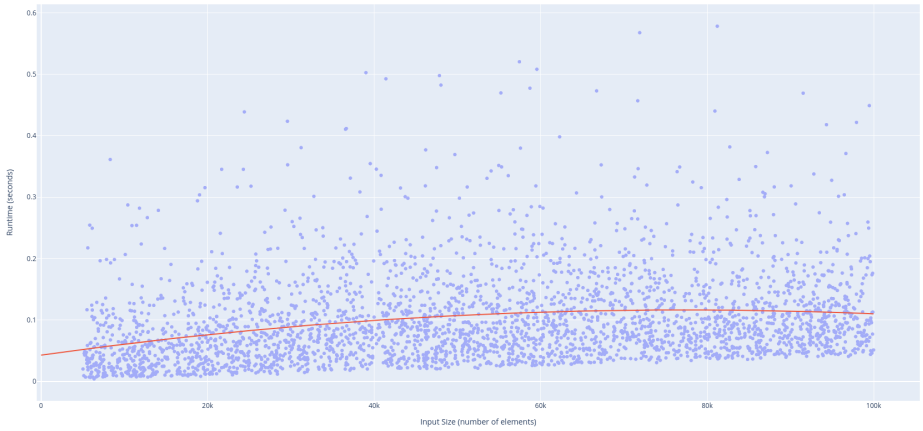

Figure 4: Merge sort runtime v/s input size plot

3 Footnotes

Code to generate graphs and this file is on github