Algorithms Lab Assignment 4

Pranav Gade

April 7, 2022

Batch: CS&AI
Roll no.: LCI2020010

1 K way merge

K way merge algorithm.

1.1 Code

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

void merge(int* arr1, int size1, int* arr2, int size2, int* arr);
void k_way_merge(int** arrays, int* sizes, int sizes_len, int*
    result);

int main(int argc, char** argv) {

    FILE* file;
    file = fopen("../k_way_merge_input.txt", "r");
    int k;
    fscanf(file, "%d\n", &k);
    int* sizes = malloc(sizeof(int) * k); // not free'd
    int** arrays = malloc(sizeof(int*) * k); // not free'd
    int total = 0;
    for (int i = 0; i < k; ++i) {
        fscanf(file, "%d\n", &sizes[i]);
        arrays[i] = malloc(sizeof(int) * sizes[i]);
        for (int j = 0; j < sizes[i]; ++j) {
            fscanf(file, "%d\n", &arrays[i][j]);
        }
        total += sizes[i];
    }

    int* result = malloc(sizeof(int) * total);

    k_way_merge(arrays, sizes, k, result);

    FILE* out;
    out = fopen("../k_way_merge_output.txt", "w");

```

```
    for (int i = 0; i < total; ++i) {
        fprintf(out, "%d\n", result[i]);
    }

    return 0;
}

void merge(int* arr1, int size1, int* arr2, int size2, int* arr) {
    // assuming arr1 and arr2 are sorted, arr3 has enough allocated space
    int k = 0;
    int i = 0;
    int j = 0;

    while (i < size1 && j < size2) {
        if (arr1[i] <= arr2[j]) {
            arr[k] = arr1[i];
            i++;
        } else {
            arr[k] = arr2[j];
            j++;
        }
        k++;
    }

    while (i < size1) {
        arr[k] = arr1[i];
        i++;
        k++;
    }

    while (j < size2) {
        arr[k] = arr2[j];
        j++;
        k++;
    }
}

void k_way_merge(int** arrays, int* sizes, int sizes_len, int* result) {
    // k-way merge arrays and put into result. assuming result has enough space
    while (sizes_len > 1) {
        for (int i = 0; i < sizes_len-1; i+=2) {
            int* arr1 = arrays[i];
            int* arr2 = arrays[i+1];
            int size1 = sizes[i];
            int size2 = sizes[i+1];
            int* res = malloc(sizeof(int) * (size1+size2)); // could optimise to be in-place
            merge(arr1, size1, arr2, size2, res);
            free(arr1);
            free(arr2);
            arrays[i/2] = res;
            sizes[i/2] = size1+size2;
        }

```

```
        sizes_len += sizes_len%2;
        sizes_len /= 2;
    }

    for (int i = 0; i < sizes[0]; ++i) {
        result[i] = arrays[0][i];
    }
    free(arrays[0]);
}
```

1.2 Input

```
4
2
1
2
2
3
4
2
7
8
2
5
6
```

1.3 Output

```
1
2
3
4
5
6
7
8
```

2 Stack reduced quicksort

Analysis of Stack reduced quicksort complexity.

2.1 Code

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

void insertion_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]);
    }

```

```c
    insertion_sort(arr, size);

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

    free(arr);
    return 0;
}

void stack_reduced_quick_sort(int *arr, int low, int high) {
    int q;
    while (low < high) {
        int pivot = high;
        int i = low - 1, j;
        for (j = low; j < high; j++) {
            if (arr[j] <= arr[pivot]) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[pivot];
        arr[pivot] = temp;
        i++;
        if (i - low < high - i) {
            stack_reduced_quick_sort(arr, low, i - 1);
            low = i + 1;
        } else {
            stack_reduced_quick_sort(arr, i + 1, high);
            high = i - 1;
        }
    }
}

void insertion_sort(int *arr, int size) {
    stack_reduced_quick_sort(arr, 0, size - 1);
}
```

2.2 Output

```
/home/p/Desktop/Programs/C/sem4_algos/cmake-build-debug/stack_reduced_quicksort 4 6 2 8 3 0 1 2
0 1 2 2 3 4 6 8
```

Figure 1: Stack reduced quicksort output

2.3 Graph

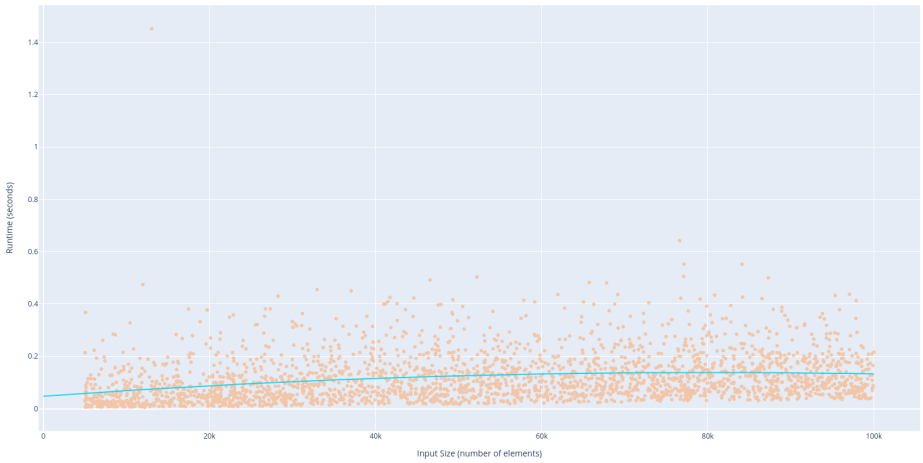

Figure 2: Stack reduced quicksort runtime v/s input size plot

3 Merge sort combined with insertion sort

Analysis of Merge sort combined with insertion sort complexity.

3.1 Code

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

void sort(int* arr, int size);
void insertion_sort(int* arr, int size);
void merge(int* arr1, int size1, int* arr2, int size2, int* arr);
void k_way_merge(int** arrays, int* sizes, int sizes_len, int*
    result);

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]);
    }

    insertion_sort(arr, size);

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

    free(arr);
    return 0;
```

```
}

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

void merge(int* arr1, int size1, int* arr2, int size2, int* arr) {
    // assuming arr1 and arr2 are sorted, arr3 has enough allocated
     space
    int k = 0;
    int i = 0;
    int j = 0;

    while (i < size1 && j < size2) {
        if (arr1[i] <= arr2[j]) {
            arr[k] = arr1[i];
            i++;
        } else {
            arr[k] = arr2[j];
            j++;
        }
        k++;
    }

    while (i < size1) {
        arr[k] = arr1[i];
        i++;
        k++;
    }

    while (j < size2) {
        arr[k] = arr2[j];
        j++;
        k++;
    }
}

void k_way_merge(int** arrays, int* sizes, int sizes_len, int*
    result) {
    // k-way merge arrays and put into result. assuming result has
    enough space
    while (sizes_len > 1) {
        for (int i = 0; i < sizes_len-1; i+=2) {
            int* arr1 = arrays[i];
            int* arr2 = arrays[i+1];
            int size1 = sizes[i];
            int size2 = sizes[i+1];
            int* res = malloc(sizeof(int) * (size1+size2)); //
    could optimise to be in-place
```

```
77              merge(arr1, size1, arr2, size2, res);
78              free(arr1);
79              free(arr2);
80              arrays[i/2] = res;
81              sizes[i/2] = size1+size2;
82          }
83
84          sizes_len += sizes_len%2;
85          sizes_len /= 2;
86      }
87
88      for (int i = 0; i < sizes[0]; ++i) {
89          result[i] = arrays[0][i];
90      }
91      free(arrays[0]);
92  }
93
94  void sort(int* arr, int size) {
95      int k = 4;
96      int* sizes = malloc(sizeof(int) * k); // not free'd
97      int** arrays = malloc(sizeof(int*) * k); // not free'd
98      int total = 0;
99      int ptr = 0;
100     for (int i = 0; i < k; ++i) {
101         sizes[i] = size/k;
102         if (i == k-1) {
103             sizes[i] += size%k;
104         }
105         arrays[i] = malloc(sizeof(int) * sizes[i]);
106         for (int j = 0; j < sizes[i]; ++j) {
107             arrays[i][j] = arr[ptr++];
108         }
109         insertion_sort(arrays[i], sizes[i]);
110         total += sizes[i];
111     }
112
113     int* result = malloc(sizeof(int) * total);
114
115     k_way_merge(arrays, sizes, k, result);
116 }
```

3.2 Output

```
/home/p/Desktop/Programs/C/sem4_algos/cmake-build-debug/merge_sort_with_insertion_sort 4 2 3 1 8 6 7 5
1 2 3 4 5 6 7 8
```

Figure 3: Merge sort combined with insertion sort output

3.3 Graph

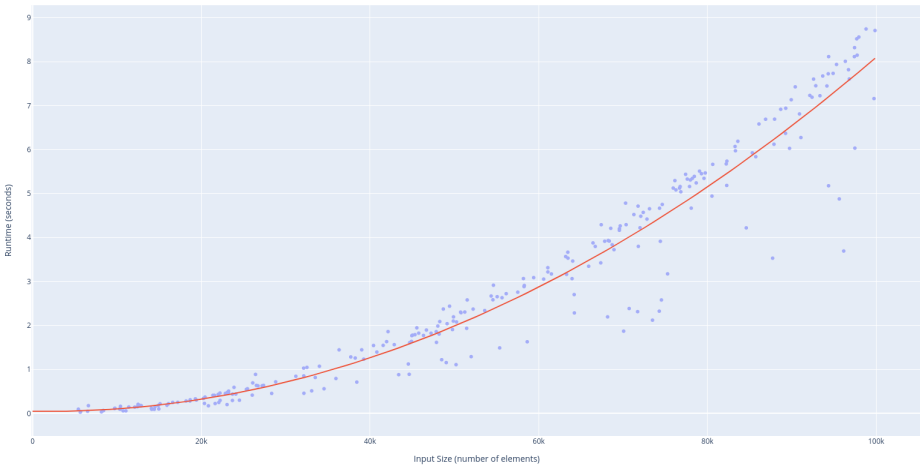

Figure 4: Merge sort combined with insertion sort runtime v/s input size plot

4 Footnotes

Code to generate graphs and this file is on github