

# 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

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 void merge(int* arr1, int size1, int* arr2, int size2, int* arr);
5 void k_way_merge(int** arrays, int* sizes, int sizes_len, int*
    result);
6
7 int main(int argc, char** argv) {
8
9     FILE* file;
10    file = fopen("../k_way_merge_input.txt", "r");
11    int k;
12    fscanf(file, "%d\n", &k);
13    int* sizes = malloc(sizeof(int) * k); // not free'd
14    int** arrays = malloc(sizeof(int*) * k); // not free'd
15    int total = 0;
16    for (int i = 0; i < k; ++i) {
17        fscanf(file, "%d\n", &sizes[i]);
18        arrays[i] = malloc(sizeof(int) * sizes[i]);
19        for (int j = 0; j < sizes[i]; ++j) {
20            fscanf(file, "%d\n", &arrays[i][j]);
21        }
22        total += sizes[i];
23    }
24
25    int* result = malloc(sizeof(int) * total);
26
27    k_way_merge(arrays, sizes, k, result);
28
29    FILE* out;
30    out = fopen("../k_way_merge_output.txt", "w");
31
```

```

32     for (int i = 0; i < total; ++i) {
33         fprintf(out, "%d\n", result[i]);
34     }
35
36     return 0;
37 }
38
39 void merge(int* arr1, int size1, int* arr2, int size2, int* arr) {
40     // assuming arr1 and arr2 are sorted, arr3 has enough allocated
    space
41     int k = 0;
42     int i = 0;
43     int j = 0;
44
45     while (i < size1 && j < size2) {
46         if (arr1[i] <= arr2[j]) {
47             arr[k] = arr1[i];
48             i++;
49         } else {
50             arr[k] = arr2[j];
51             j++;
52         }
53         k++;
54     }
55
56     while (i < size1) {
57         arr[k] = arr1[i];
58         i++;
59         k++;
60     }
61
62     while (j < size2) {
63         arr[k] = arr2[j];
64         j++;
65         k++;
66     }
67 }
68
69 void k_way_merge(int** arrays, int* sizes, int sizes_len, int*
    result) {
70     // k-way merge arrays and put into result. assuming result has
    enough space
71     while (sizes_len > 1) {
72         for (int i = 0; i < sizes_len-1; i+=2) {
73             int* arr1 = arrays[i];
74             int* arr2 = arrays[i+1];
75             int size1 = sizes[i];
76             int size2 = sizes[i+1];
77             int* res = malloc(sizeof(int) * (size1+size2)); //
    could optimise to be in-place
78             merge(arr1, size1, arr2, size2, res);
79             free(arr1);
80             free(arr2);
81             arrays[i/2] = res;
82             sizes[i/2] = size1+size2;
83         }
84     }

```

```

85     sizes_len += sizes_len%2;
86     sizes_len /= 2;
87 }
88
89 for (int i = 0; i < sizes[0]; ++i) {
90     result[i] = arrays[0][i];
91 }
92 free(arrays[0]);
93 }

```

## 1.2 Input

```

1 4
2 2
3 1
4 2
5 2
6 3
7 4
8 2
9 7
10 8
11 2
12 5
13 6

```

## 1.3 Output

```

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

```

# 2 Stack reduced quicksort

Analysis of Stack reduced quicksort complexity.

## 2.1 Code

```

1 #include <stdlib.h>
2 #include <stdio.h>
3
4 void insertion_sort(int *arr, int size);
5
6 int main(int argc, char **argv) {
7     int size = argc - 1;
8     int *arr = malloc(sizeof(int) * size);
9     for (int i = 1; i <= size; ++i) {
10         arr[i - 1] = atoi(argv[i]);
11     }
12 }

```

```

13     insertion_sort(arr, size);
14
15     for (int i = 0; i < size; ++i) {
16         printf("%d ", arr[i]);
17     }
18
19     free(arr);
20     return 0;
21 }
22
23 void stack_reduced_quick_sort(int *arr, int low, int high) {
24     int q;
25     while (low < high) {
26         int pivot = high;
27         int i = low - 1, j;
28         for (j = low; j < high; j++) {
29             if (arr[j] <= arr[pivot]) {
30                 i++;
31                 int temp = arr[i];
32                 arr[i] = arr[j];
33                 arr[j] = temp;
34             }
35         }
36         int temp = arr[i + 1];
37         arr[i + 1] = arr[pivot];
38         arr[pivot] = temp;
39         i++;
40         if (i - low < high - i) {
41             stack_reduced_quick_sort(arr, low, i - 1);
42             low = i + 1;
43         } else {
44             stack_reduced_quick_sort(arr, i + 1, high);
45             high = i - 1;
46         }
47     }
48 }
49
50 void insertion_sort(int *arr, int size) {
51     stack_reduced_quick_sort(arr, 0, size - 1);
52 }

```

## 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

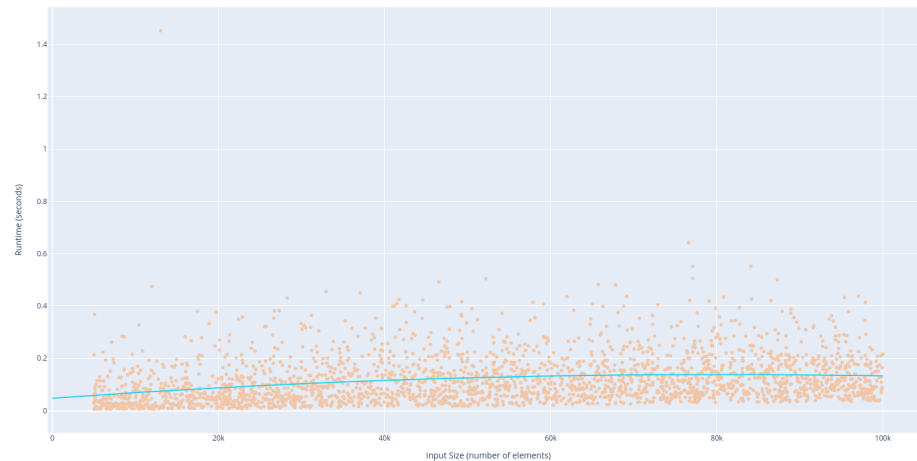


Figure 2: Stack reduced quicksort runtime v/s input size plot

## 3 Merge sort combined with insertion sort

Analysis of Merge sort combined with insertion sort complexity.

### 3.1 Code

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 void sort(int* arr, int size);
5 void insertion_sort(int* arr, int size);
6 void merge(int* arr1, int size1, int* arr2, int size2, int* arr);
7 void k_way_merge(int** arrays, int* sizes, int sizes_len, int*
   result);
8
9 int main(int argc, char** argv) {
10     int size = argc - 1;
11     int* arr = malloc(sizeof(int) * size);
12     for (int i = 1; i <= size; ++i) {
13         arr[i-1] = atoi(argv[i]);
14     }
15
16     insertion_sort(arr, size);
17
18     for (int i = 0; i < size; ++i) {
19         printf("%d ", arr[i]);
20     }
21
22     free(arr);
23     return 0;
```

```

24 }
25
26 void insertion_sort(int* arr, int size) {
27     for (int i = 0; i < size; ++i) {
28         int curr = arr[i];
29         int j = i-1;
30         while (arr[j] > curr && j >= 0) {
31             arr[j+1] = arr[j];
32             j--;
33         }
34         arr[j+1] = curr;
35     }
36 }
37
38 void merge(int* arr1, int size1, int* arr2, int size2, int* arr) {
39     // assuming arr1 and arr2 are sorted, arr3 has enough allocated
    space
40     int k = 0;
41     int i = 0;
42     int j = 0;
43
44     while (i < size1 && j < size2) {
45         if (arr1[i] <= arr2[j]) {
46             arr[k] = arr1[i];
47             i++;
48         } else {
49             arr[k] = arr2[j];
50             j++;
51         }
52         k++;
53     }
54
55     while (i < size1) {
56         arr[k] = arr1[i];
57         i++;
58         k++;
59     }
60
61     while (j < size2) {
62         arr[k] = arr2[j];
63         j++;
64         k++;
65     }
66 }
67
68 void k_way_merge(int** arrays, int* sizes, int sizes_len, int*
    result) {
69     // k-way merge arrays and put into result. assuming result has
    enough space
70     while (sizes_len > 1) {
71         for (int i = 0; i < sizes_len-1; i+=2) {
72             int* arr1 = arrays[i];
73             int* arr2 = arrays[i+1];
74             int size1 = sizes[i];
75             int size2 = sizes[i+1];
76             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

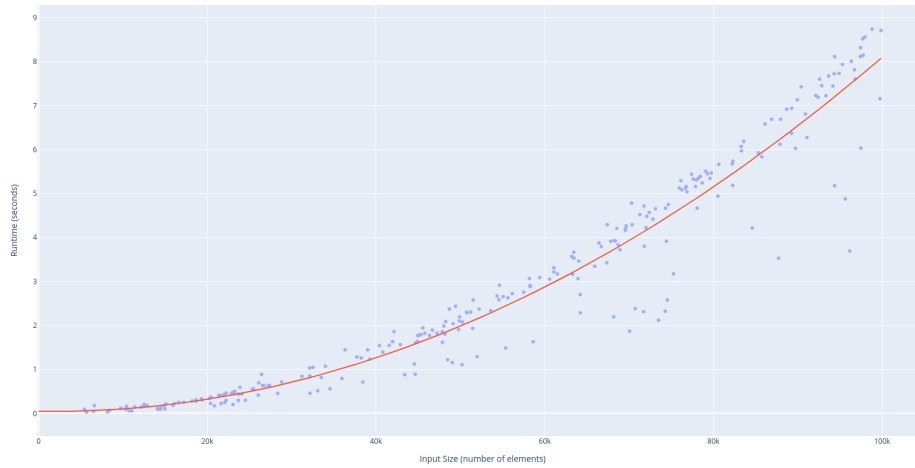


Figure 4: Merge sort combined with insertion sort runtime v/s input size plot

## 4 Footnotes

Code to generate graphs and this file is on [github](#)