Algorithms Lab Assignment 2

Pranav Gade

February 10, 2022

Batch:	CS&AI
Roll no.:	LCI2020010

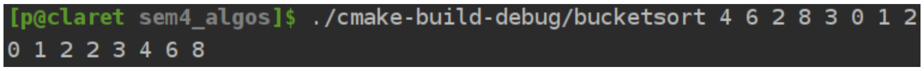
1 Bucket sort

Analysis of Bucket sort complexity.

1.1 Code

```c
#include <stdlib.h>
#include <stdio.h>
#include <limits.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 insertionsort(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 sort(int* arr, int size) {
    int buckets[16][size];
    int sizes[16] = {0};

    int max = INT_MIN;
    for (int i = 0; i < size; ++i) {
        if (arr[i] > max) max = arr[i];
    }
    int bit = 0;
    while (max > 0) {
        max /= 2;
        bit++;
    }
    bit -= 4;
    if (bit < 0) bit = 0;
    for (int i = 0; i < size; ++i) {
        int pos = (arr[i] & (0xf << bit)) >> bit;
        buckets[pos][sizes[pos]++] = arr[i];
    }

    for (int i = 0; i < 16; ++i) insertionsort(buckets[i], sizes[i]);

    int pos = 0;
    int ptr = 0;
    for (int i = 0; i < size; ++i) {
        while (ptr >= sizes[pos]) {
            pos++;
            ptr = 0;
        }
        arr[i] = buckets[pos][ptr++];
    }
}
```

1.2 Output



```
[p@claret sem4_algos]$ ./cmake-build-debug/bucketsort 4 6 2 8 3 0 1 2
0 1 2 2 3 4 6 8
```

Figure 1: Bucket sort test output

1.3 Graph

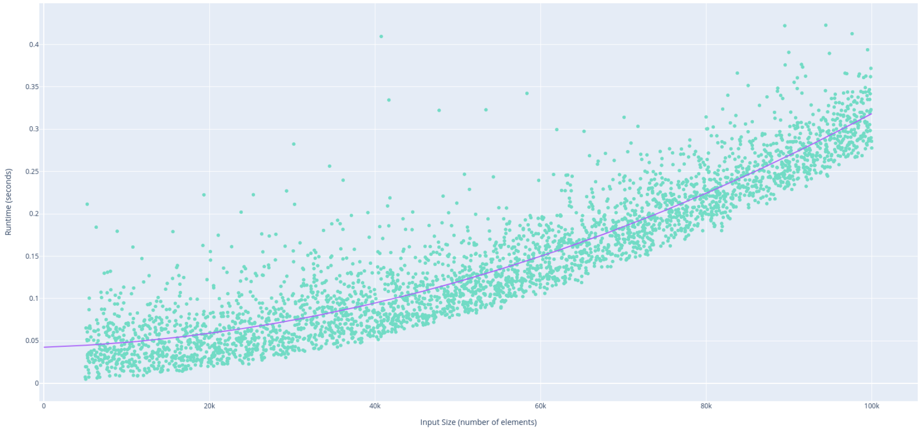


Figure 2: Bucket sort runtime v/s input size plot

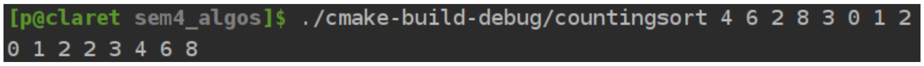
2 Counting sort

Analysis of Counting sort complexity.

2.1 Code

```c
#include <stdlib.h>
#include <stdio.h>
#include <limits.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 sort(int* arr, int size) {
    int max = INT_MIN;
    int min = INT_MAX;
    for (int i = 0; i < size; ++i) {
        if (arr[i] > max) max = arr[i];
        if (arr[i] < min) min = arr[i];
    }

    int len = max - min + 2;
    int counts[len];
    for (int i = 0; i < len; ++i) counts[i] = 0;
    for (int i = 0; i < size; ++i) counts[arr[i] - min]++;

    int ptr = 0;
    for (int i = 0; i < size; ++i) {
        while (counts[ptr] <= 0) ptr++;
        arr[i] = ptr;
        counts[ptr]--;
    }
}
```

2.2 Output



```
[p@claret sem4_algos]$ ./cmake-build-debug/countingsort 4 6 2 8 3 0 1 2
0 1 2 2 3 4 6 8
```

Figure 3: Counting sort test output

2.3 Graph

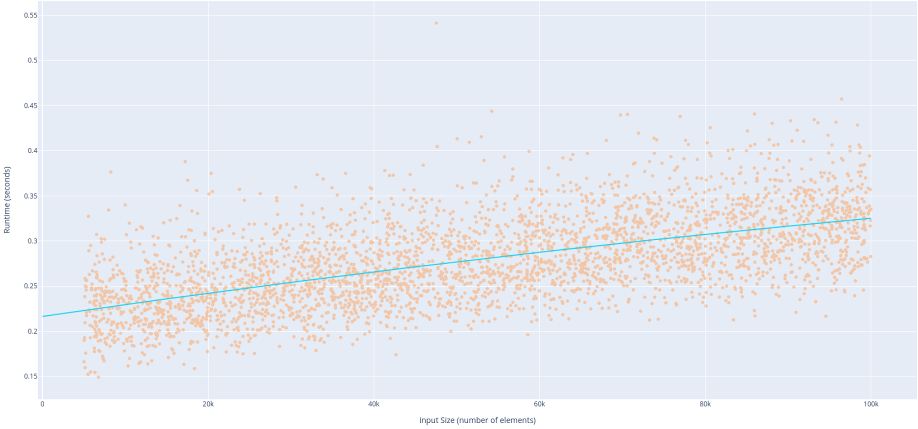


Figure 4: Counting sort runtime v/s input size plot

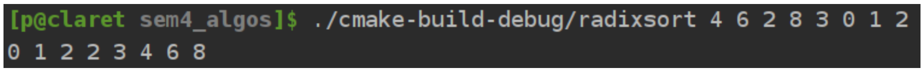
3 Radix sort

Analysis of Radix sort complexity.

3.1 Code

```c
#include <stdlib.h>
#include <stdio.h>
#include <limits.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 sort_helper(int* arr, int size, int bit) {
    int buckets[16][size];
    int sizes[16] = {0};
    for (int i = 0; i < size; ++i) {
        int pos = (arr[i] & (0xf << bit)) >> bit;
        buckets[pos][sizes[pos]++] = arr[i];
    }
    int pos = 0;
    int ptr = 0;
    for (int i = 0; i < size; ++i) {
        while (ptr >= sizes[pos]) {
            pos++;
            ptr = 0;
        }
        arr[i] = buckets[pos][ptr++];
    }
}

void sort(int* arr, int size) {
    int max = INT_MIN;
    for (int i = 0; i < size; ++i) {
        if (arr[i] > max) max = arr[i];
    }
    int bit = 0;
    while (max > 0) {
        max /= 2;
        bit++;
    }
    bit -= 4;
    if (bit < 0) bit = 0;

    int ptr = 0;
    while (bit >= ptr) {
        sort_helper(arr, size, ptr);
        ptr += 4;
    }
}
```

3.2 Output



```
[p@claret sem4_algos]$ ./cmake-build-debug/radixsort 4 6 2 8 3 0 1 2
0 1 2 2 3 4 6 8
```

Figure 5: Radix sort test output

3.3 Graph

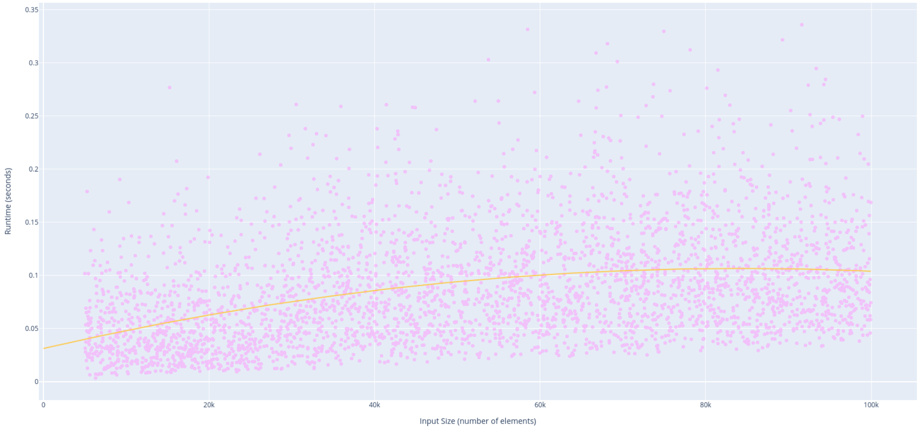


Figure 6: Radix sort runtime v/s input size plot

4 Footnotes

Code to generate graphs and this file is on github