

# Assignment 1: Introduction to Systems Programming

Francis Vo

October 9, 2012

## 1 Mathematical Analysis

### 1.1 Algorithm 1

**Data:** Integer array  $A$  of size  $N$

**Result:** Greatest Sum of Subarray

```
1 for  $i \leftarrow 0$  to  $N$  do
2   for  $j \leftarrow i$  to  $N$  do
3      $s \leftarrow 0$ 
4     for  $k \leftarrow i$  to  $j$  do
5        $s \leftarrow s + A[k]$ 
6     end
7     if  $s > max$  then
8        $max \leftarrow s$ 
9     end
10  end
11 end
```

**Algorithm 1:** Pseudocode for Basic Enumeration

### 1.2 Algorithm 2

**Data:** Integer array  $A$  of size  $N$

**Result:** Greatest Sum of Subarray

```
1 for  $i \leftarrow 0$  to  $N$  do
2    $s \leftarrow 0$ 
3   for  $j \leftarrow i$  to  $N$  do
4      $s \leftarrow A[j]$ 
5     if  $s > max$  then
6        $max \leftarrow s$ 
7     end
8   end
9 end
```

**Algorithm 2:** Pseudocode for Better Enumeration

## **2 Theoretical Correctness**

## **3 Testing**

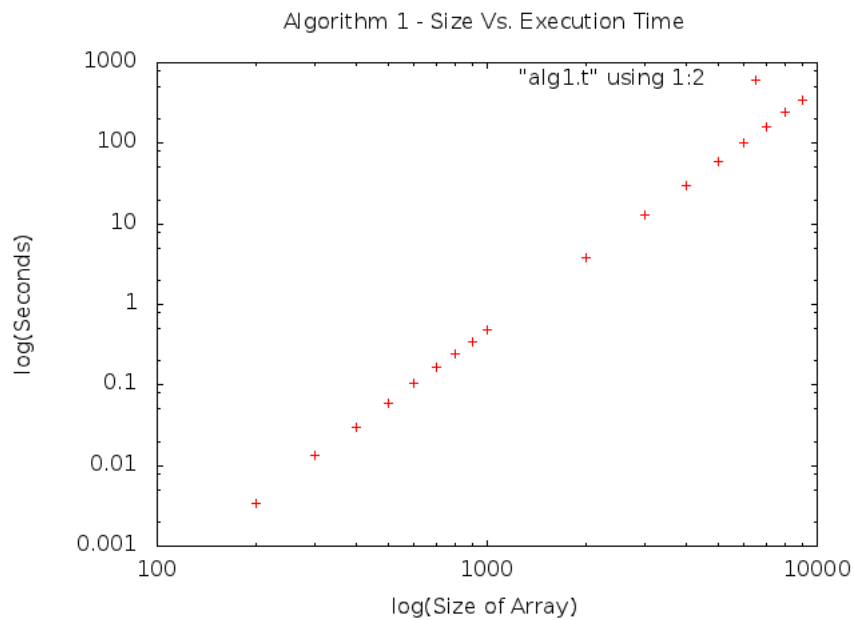
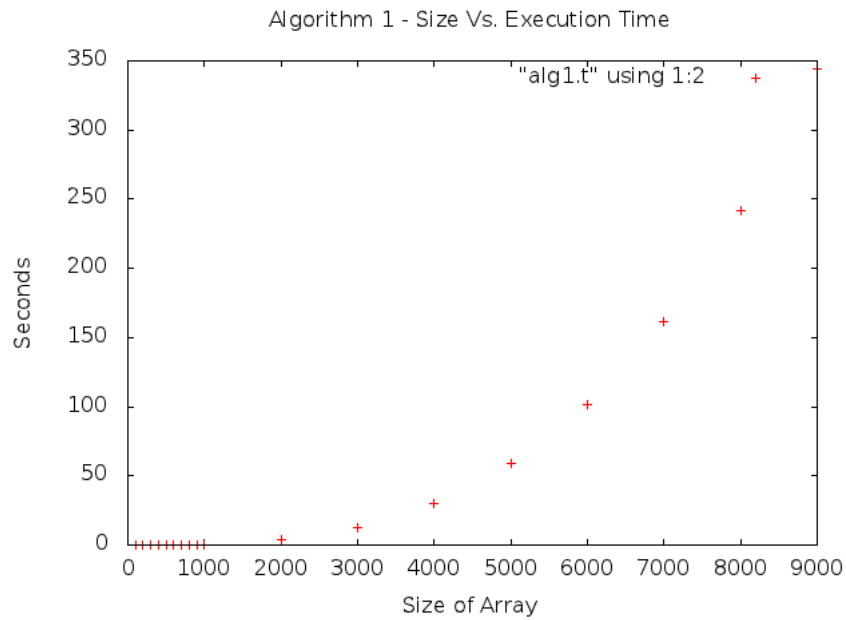
931678074 - 5703

930569466 - 8184

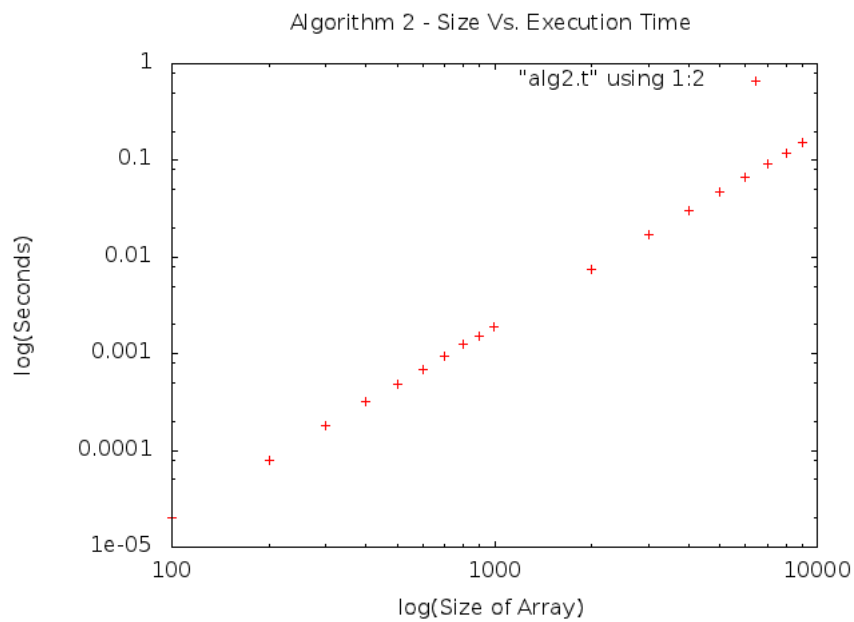
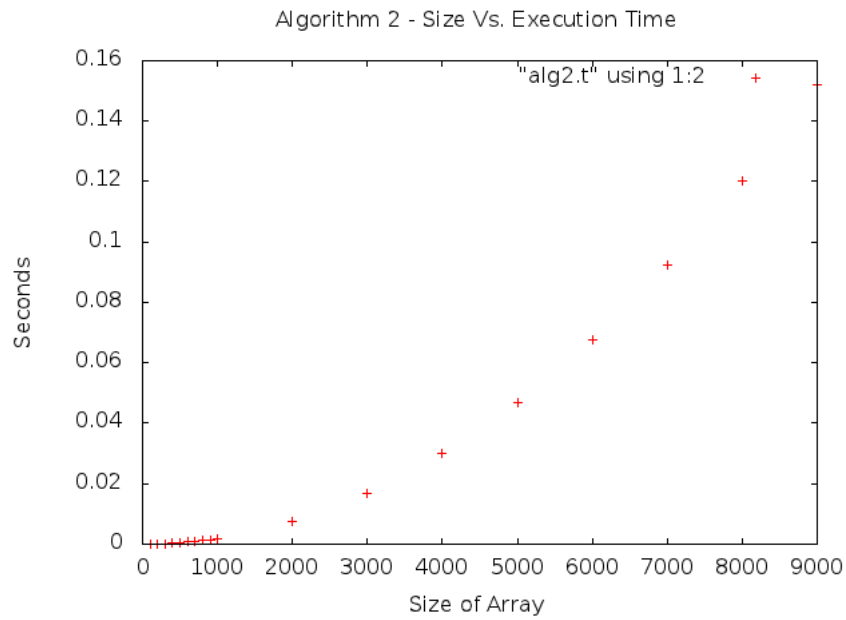
932086449 - 4949

## 4 Experimental Analysis

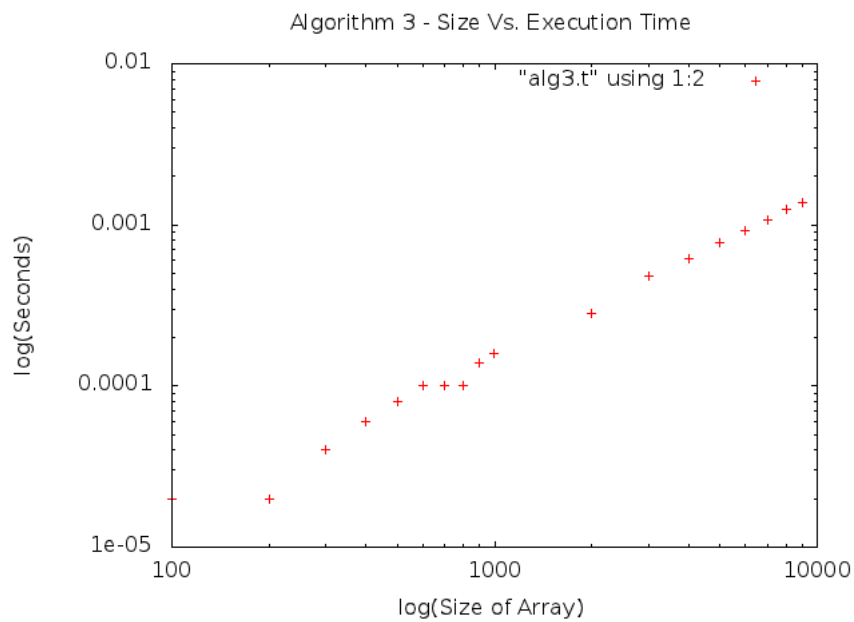
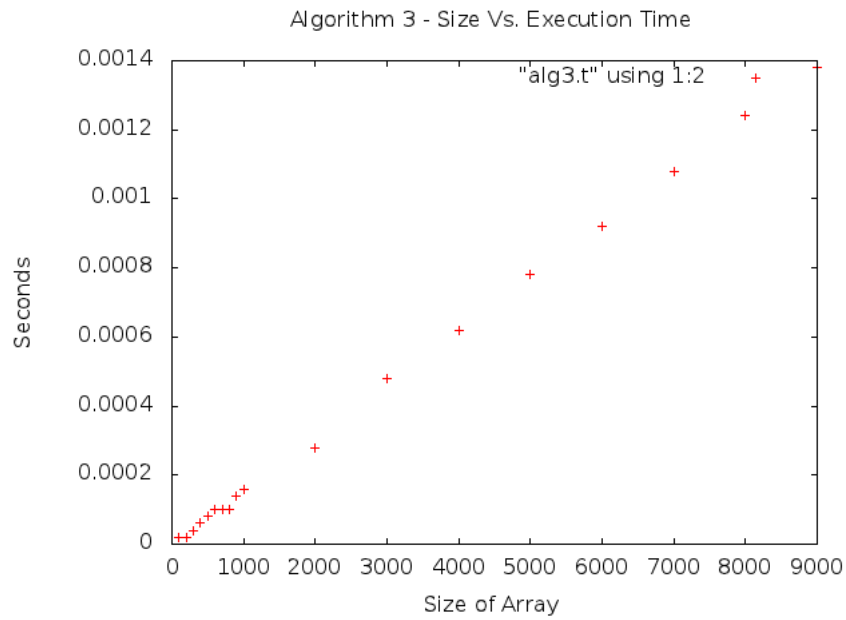
### 4.1 Algorithm 1



## 4.2 Algorithm 2



### 4.3 Algorithm 3



## 5 Extrapolation and Interpretation

### 5.1 Algorithm 3

$$\begin{aligned} size &= 6462453 \times time + 21.41728 \\ time = 3600 &\rightarrow size = 2.3 \times 10^{10} \end{aligned}$$

## 6 Code

### 6.1 Algorithm 1

```
1  /*
2  * Enumeration
3  * Loop over each pair of indices i; j and compute the sum from k=i
4  *   to j of a[k].
5  * Keep the best sum you have found so far.
6  */
7  using namespace std;
8
9
10 int MaxSubarray(int a[], int n){
11
12     int i,j,k;
13     int max = a[0];
14     int sum;
15     for(i = 0; i < n; ++i ){
16         for (j = i; j < n; ++j){
17             sum = 0;
18             for (k = i; k <=j; ++k){
19                 sum += a[k];
20             }
21             if(max < sum){
22                 max = sum;
23             }
24         }
25     }
26     return max;
27 }
```

alg1.cpp

## 6.2 Algorithm 2

```
1  /*
2  * Better Enumeration
3  * Notice that in the previous algorithm, the same sum is computed
4  * many times.
5  * In particular, notice that sum from k=i to j of a[k] can be
6  * computed from sum from k=i to j - 1 of a[k] in O(1) time,
7  * rather than starting from scratch.
8  * Write a new version of the first algorithm that takes advantage
9  * of this observation.
10 */
11
12 using namespace std;
13
14 int MaxSubarray(int a[], int n){
15     int i,j,k;
16     int max = a[0];
17     int sum;
18     for(i = 0; i < n; ++i){
19         sum = 0;
20         for(j = i; j < n; ++j){
21             sum += a[j];
22             if(max < sum){
23                 max = sum;
24             }
25         }
26     }
27     return max;
28 }
```

alg2.cpp



### 6.3 Algorithm 3

```
1  /*
2  * Divide and Conquer
3  * If we split the array into two halves, we know that the maximum
   subarray will either be
4  *     * contained entirely in the first half,
5  *     * contained entirely in the second half, or
6  *     * made of a suffix of the first half of maximum sum and a
   prefix of the second half of maximum sum
7  * The first two cases can be found recursively. The last case can
   be found in linear time.
8  */
9
10 using namespace std;
11
12 void MaxSubarray_h(int a[], int n, int b[]) {
13     if (n <= 1) {
14         b[0] = a[0];
15         b[1] = a[0];
16         b[2] = a[0];
17         b[3] = a[0];
18         return;
19     }
20     int i = n/2;
21     int *left = new int[4];
22     MaxSubarray_h(a, i, left);
23     int *right = new int[4];
24     MaxSubarray_h(a+i, n-i, right);
25
26     int sum = left[0] + right[0];
27     int l = left[0] + right[1];
28     int r = left[2] + right[0];
29     int mid = left[2] + right[1];
30
31     l = l > left[1] ? l : left[1];
32     r = r > right[2] ? r : right[2];
33     int m = left[3] > right[3] ? left[3] : right[3];
34     m = m > mid ? m : mid;
35
36     b[0] = sum;
37     b[1] = l;
38     b[2] = r;
39     b[3] = m;
40 }
41
42
43 int MaxSubarray(int a[], int n) {
44     int *p = new int[4];
45     MaxSubarray_h(a, n, p);
46     int s1 = p[0] > p[1] ? p[0] : p[1];
47     int s2 = p[2] > p[3] ? p[2] : p[3];
48     s1 = s1 > s2 ? s1 : s2;
49     return s1;
50 }
```

alg3.cpp