

Assignment 1: Introduction to Systems Programming

Francis Vo

October 2, 2012

- 1 Mathematical Analysis
 - 2 Theoretical Correctness
 - 3 Testing
 - 4 Experimental Analysis
 - 5 Extrapolation and Interpretation
 - 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  #include <iostream>
8  #include <cstdio>
9  #include <stdio.h>
10 #include <stdlib.h>
11
12 int MaxSubarray(int a[], int n);
13
14 int main(int argc, char **argv){
15     int input[] = {31, -41, 59, 26, -53, 58, 97, -93, -23, 84};
16     std::cout << MaxSubarray(input,10) << std::endl;
17 }
18
19 int MaxSubarray(int a[], int n){
20
21     int i,j,k;
22     int max = a[0];
23     int sum;
24     for(i = 0; i < n; ++i){
25         for (j = i; j < n; ++j){
```

```

26     sum = 0;
27     for (k = i; k <=j; ++k){
28         sum += a[k];
29     }
30     if(max < sum){
31         max = sum;
32     }
33 }
34 }
35 return max;
36 }

```

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 #include <iostream>
13 #include <cstdio>
14 #include <stdio.h>
15 #include <stdlib.h>
16
17 int MaxSubarray(int a[], int n);
18
19 int main(int argc, char **argv){
20     int input[] = {31, -41, 59, 26, -53, 58, 97, -93, -23, 84};
21     std::cout << MaxSubarray(input,10) << std::endl;
22 }
23
24 int MaxSubarray(int a[], int n){
25     int i,j,k;
26     int max = a[0];
27     int sum;
28     for(i = 0; i < n; ++i ){
29         sum = 0;
30         for (j = i; j < n; ++j){
31             sum += a[j];
32             if(max < sum){
33                 max = sum;
34             }
35         }
36     }
37     return max;
38 }

```

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
4    subarray will either be
5  *      * contained entirely in the first half ,
6  *      * contained entirely in the second half , or
7  *      * made of a suffix of the first half of maximum sum and a
8    prefix of the second half of maximum sum
9  * The first two cases can be found recursively . The last case can
10   be found in linear time .
11 */
```

alg3.cpp