Algorithms

September 2019

Degree in Computer Science Engineering

Practical 1

Submission deadline: Tuesday, 10 October

Maximum subsequence sum: Given n integer numbers $a_1, a_2, ..., a_n$, find the maximum value of $\sum_{k=i}^{j} a_k$ with $1 \leq i \leq j \leq n$ (for convenience, the sum of the maximum subsequence is 0 if all the integers are negative).

We propose two algorithms to solve this problem:

Algorithm 1: $O(n^2)$

```
function maxSubSum1 (v[1..n])
    MaxSum := 0;
    for i := 1 to n do
        ThisSum := 0 ;
        for j := i to n do
            ThisSum := ThisSum + v[j]
            if ThisSum > MaxSum then
                MaxSum := ThisSum
            end if
        end for
    end for ;
    return MaxSum
end function
```

Algorithm 2: $O(n)$

```
function maxSubSum2 (v[1..n])
    ThisSum := 0; MaxSum :=0;
    for j := 1 to n do
        ThisSum := ThisSum + v[j] ;
        if ThisSum > MaxSum then
            MaxSum := ThisSum
        else if ThisSum < 0 then
            ThisSum := 0
        end if
    end for ;
    return MaxSum
end function
```

You must:

1. Implement the proposed algorithms in C (see Figure 1).
2. Validate that the algorithms work correctly. Test them on the following sequences:

sequence	result
-9, 2, -5, -4, 6	6
4, 0, 9, 2, 5	20
-2, -1, -9, -7, -1	0
9, -2, 1, -7, -8	9
15, -2, -5, -4, 16	20
7, -5, 6, 7, -7	15

```
#include <stdio.h>
int maxSubSum1(int v[], int n) {
  int i, j;
  int thisSum, maxSum = 0;
  for (i = 0; i < n; i++) {
    thisSum = 0;
    for (j = i; j < n; j++) {
      thisSum += v[j];
      if (thisSum > maxSum) {
        maxSum = thisSum;
      }
    }
  }
  return maxSum;
}
int maxSubSum2(int v[], int n) {
/* ... */
}
void print_array(int v[], int n){
/* ... */
}
void test_1() {
/* ... */
}
void test2() {
  int i, a, b;
  int v[9];
  printf("test\n");
  printf("%33s%15s%15s\n", "", "maxSubSum1", "maxSubSum2");
  for (i=0; i<10; i++) {
    random_init(v,9);
    print_array(, 9);
    a = maxSubSum1(v, 9);
    b = maxSubSum2(v, 9);
    printf("%15d%15d\n", a, b);
  }
}

int main() {
  init_seed();
  test1();
  test2();
  return 0;
}
```

Figure 1: Code with the `maxSubSum1` function and the second test

Additionally, perform a second test with randomly-generated arrays (Figure 2) checking that both algorithms return the same result (Figure 3).

```
void init_seed() {
  srand(time(NULL));
  /* set the seed of a new sequence of pseudo-random integers */
}
void random_init(int v [], int n) {
  int i, m=2*n+1;
  for (i=0; i < n; i++)
    v[i] = (rand() % m) - n;
  /* generate pseudo-random numbers between -n and +n */
}
```

Figure 2: Initialization of an array with pseudo-random integers in the range $[-n,\ldots,+n]$

```
                                        maxSubSum1    maxSubSum2
[ 7   0 -3  3 -1  1 -5  5 -8 ]              7             7
[ -8  8  1 -1  7  6 -5 -4 -7 ]             21            21
[ -1 -7 -8 -6  7 -2  1 -2 -7 ]              7             7
[  8 -3  8  9 -9 -5 -4  3  1 ]             22            22
[  3  8 -4  5  6 -9 -4 -8  7 ]             18            18
[  9 -6 -6  8  2 -7 -9  5  9 ]             14            14
[ -5 -7 -6 -3 -8 -3 -5 -9 -3 ]              0             0
[ -3 -7 -9  7  6  5 -7 -2 -2 ]             18            18
[ -7  0  0  7 -1  3 -9 -5 -8 ]              9             9
[  9 -5  3  8 -1  6  9  3  4 ]             36            36
```

Figure 3: Applied on different random arrays, both functions return the same results.

3. For each of the two algorithms, determine the execution times with random arrays of size n, with n equalling $500, 1000, 2000, 4000, 8000, 16000$, and 32000. Use the code in Figure 4 to obtain the system time. To generate the test data, use the code in Figure 2, which generates arrays of pseudo-random numbers in the range $[-n,\ldots,n]$.

4. Analyse the obtained results by means of an empirical verification of the theoretical complexity (Figure 5). You will also need to perform an empirical check using an underestimated and an overestimated bound for each algorithm.

5. Place the `C` source files and the report with the empirical study of complexity at a P1 folder, using SVN in **https://svn.fic.udc.es/grao2/alg/19-20/john.smith** (replace **john.smith** by your login).

```
#include <sys/time.h>
double microseconds() {    /* obtains the system time in microseconds */
  struct timeval t;
  if (gettimeofday(&t, NULL) < 0 )
    return 0.0;
  return (t.tv_usec + t.tv_sec * 1000000.0);
}
```

Figure 4: Obtaining the system time

```
$ ./p1
MaxSubSum 1
           n           t(n)     t(n)/n^1.8       t(n)/n^2     t(n)/n^2.2
(*)      500        341.666       0.004736       0.001367       0.000394
        1000       1200.000       0.004777       0.001200       0.000301
        2000       4819.000       0.005509       0.001205       0.000263
        4000      19178.000       0.006296       0.001199       0.000228
        8000      85178.000       0.008031       0.001331       0.000221
       16000     332606.000       0.009006       0.001299       0.000187
       32000    1270463.000       0.009879       0.001241       0.000156
```

Figure 5: Part of a possible output to screen by the program that measures the execution times of the first algorithm.