Algorithms 2024/25

Degree in Computer Science Engineering

Practical 2

Insertion sort and quicksort

Submission deadline: Saturday, 19 October, 23:59

The problem consists in sorting an array of n integers in ascending order. The sorting algorithms used will be *insertion sort* and *quick sort* with random pivot selection:

```
procedure Insertion Sort (var v[1..n])
  for i := 2 to n do
    x := v[i] ;
    j := i-1 ;
    while j > 0 and v[j] > x do
      v[j+1] := v[j] ;
      j := j-1
    end while ;
    v[j+1] := x
  end for
end procedure

procedure Quicksort (v[1..n])
  Quicksort Auxiliary (v[1..n])
end procedure

procedure Quicksort Auxiliary (v[left..right])
  if left < right then
    x := {random number in the range [left..right]};
    pivot := v[x];
    swap(v[left], v[x]) ;
    i := left + 1 ;
    j := right ;
    while i <= j do
      while i <= right and v[i] < pivot do i := i + 1 end while ;
      while v[j] > pivot do j := j - 1 end while ;
      if i <= j then
        swap (v[i], v[j]) ;
        i := i + 1 ;
        j := j - 1
      end if
    end while ;
    swap (v[left], v[j]) ;
    Quick Sort Auxiliary (v[left..j-1]) ;
    Quick Sort Auxiliary (v[j+1..right])
  end if
end procedure
```

1. Implement the insertion sort and quicksort algorithms.

```
void ins_sort (int v [], int n) {
  /* ... */
}

void quick_sort_aux (int v [], int left, int right) {
  /* ... */
}

void quick_sort (int v [], int n) {
  quick_sort_aux(v, 0, n-1);
}
```

2. Validate that both sorting methods work correctly (figure 1).

```
> ./test
Insertion sort with random initialization
3, -3, 0, 17, -5, 2, 11, 13, 6, 1, 7, 14, 1, -2, 5, -14, -2
sorted? 0
sorting...
-14, -5, -3, -2, -2, 0, 1, 1, 2, 3, 5, 6, 7, 11, 13, 14, 17
sorted? 1

Insertion sort with descending initialization
10, 9, 8, 7, 6, 5, 4, 3, 2, 1
sorted? 0
sorting...
1, 2, 3, 4, 5, 6, 7, 8, 9, 10
sorted? 1
```

Figure 1: Possible test for insertion sort

3. For both insertion sort and quicksort, calculate the execution times for different values of n and for three different initial conditions: (a) the array is already sorted in ascending order, (b) the array is already sorted in descending order, and (c) the array is initially unsorted (see figure 2).

4. Analyze the obtained times by empirically determining the complexity of the algorithms for each of the different initial conditions of the array (i.e., 6 tables) (see figure 3).

5. Submit the files with the C code and the `.txt` file with the report through the task *Practical 2 Submission* on the Algorithms page in the virtual campus. Remember that the deadline to complete the task is Saturday, October 19 at 23:59, and once the files are uploaded, they cannot be changed. All team members must submit the assignment.

```
#include <stdlib.h>
void init_seed() {
  srand(time(NULL));
}
void random_init(int v [], int n) {/* we generate pseudo-random numbers between -n y +n */
  int i, m=2*n+1;
  for (i=0; i < n; i++)
    v[i] = (rand() % m) - n;
}
void ascending_init(int v [], int n) {
  int i;
  for (i=0; i < n; i++)
    v[i] = i;
}
```

Figure 2: Random and ascending initialization

```
Insertion sort with descending initialization
           n           t(n)     t(n)/n^1.8       t(n)/n^2     t(n)/n^2.2
(*)      500         247.03       0.003425       0.000988       0.000285
        1000         953.00       0.003794       0.000953       0.000239
        2000        3818.00       0.004365       0.000955       0.000209
        4000       15471.00       0.005079       0.000967       0.000184
        8000       69474.00       0.006550       0.001086       0.000180
       16000      257089.00       0.006961       0.001004       0.000145
       32000     1023540.00       0.007959       0.001000       0.000126
```

Figure 3: Part of the possible output to screen from running the main program