Algorithms 2021/22

Degree in Computer Science Engineering

Practical 3

Submission deadline: Saturday, 20th November 2021, at 23:59

Insertion sort and quicksort

Insertion sort

Insertion sort algorithm:

```
procedure insertionSort (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
```

You must:

1. Implement the insertion sort algorithm.

   ```
   void ins_sort (int v [], int n);
   ```

2. Validate that the implementation works correctly.

   ```
   > ./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

   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
   ```

3. Determine the execution times for different array sizes and for three different initial situations: (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.

4. Empirically calculate the complexity of the algorithm for each of the different initial situations of the array (figure 1).

```
Insertion sort with descending initialization
           n           t(n)     t(n)/n^1.8       t(n)/n^2     t(n)/n^2.2
(*)      500         357.324        0.004954        0.001429        0.000412
        1000        1577.000        0.006278        0.001577        0.000396
        2000        6103.000        0.006977        0.001526        0.000334
        4000       23603.000        0.007749        0.001475        0.000281
        8000       92347.000        0.008707        0.001443        0.000239
       16000      361434.000        0.009786        0.001412        0.000204
       32000     1446534.000        0.011248        0.001413        0.000177
```

Figura 1: Part of the possible output to screen of the main program's execution

Quicksort

Quicksort algorithm with median of three pivot selection and a threshold to detect small arrays:

```
procedure median3 (V[i..j])
  k := (i + j) div 2 ;                  /* precondition: i < j */
  if V[k] > V[j] then swap (V[k], V[j]) ;
  end if
  if V[k] > V[i] then swap (V[k], V[i]) ;
  end if
  if V[i] > V[j] then swap (V[i], V[j]) ;
  end if
end procedure

procedure sortAux (V[left..right])
  if left+THRESHOLD <= right then       /* THRESHOLD >= 1 */
    median3 (V[left..right]) ;           /* the pivot is at 'left' and at 'right' there will be */
                                        /* a value greater or equal than the pivot */
    pivot := V[left] ;
    i := left ;
    j := right ;
    repeat
      repeat i := i + 1 ; until V[i] >= pivot ;
      repeat j := j - 1 ; until V[j] <= pivot ;
      swap (V[i], V[j]);
    until j <= i ;
    swap (V[i], V[j]) ;         /* undo last swap */
    swap (V[left], V[j]) ;
    sortAux (V[left..j-1]);
    sortAux (V[j+1..right])
  end if
end procedure

procedure quickSort (V[1..n])
  sortAux(V[1..n]);
  if (THRESHOLD > 1) then
    insertionSort (V[1..n])
  end if
end procedure
```

You must:

1. Implement the quicksort algorithm.

```
void quick_sort(int v [], int n) {
  sort_aux(v, 0, n-1);
  if (THRESHOLD > 1)
    ins_sort(v, n);
}
```

2. Validate that the implementation works correctly (with threshold = 1).
3. Run the algorithm on arrays of different sizes and initial situations (ascendingly or descendingly sorted, or unsorted), and with different values for the threshold: 1 (insertion sort is never called), 10 and 100.
4. Compare the times obtained for each threshold used. As a function of the initial situation of the array, what threshold produces the best times? Why?
5. Empirically calculate the complexity of the algorithm for each of the different initial situations of the array and each of the thresholds (i.e., 9 tables).

Submit the files with the C source code and the `.txt` file with the report by means of the task *Practical 3 Submission* in the Algorithms website at `https://campusvirtual.udc.gal`. Remember that the deadline to complete the task expires on Saturday, 20th November, at 23:59. Once uploaded, files cannot be changed. **The work has to be submitted by all the members of each team.**