CSC2059 Data Structures and Algorithms

Practical 9: Sorting

Wednesday 8th December 2021

Launch Visual Studio. Create a new Solution Practical9 and name the first project Sort. In project Sort1 create a source file test.cpp.

Project Sort. Testing sorting algorithms

(1) Testing if an array is sorted

In your main method, create an array of N sorted integers, something like:

```
for (int i = 0; i < N; i++)
       a[i] = 3 * i + 1;
```

and an array of N unsorted integers, something like:

```
for (int i = 0; i < N; i++)
       a[i] = rand() % 1000;
```

Now write a function

```
bool isSorted(int* a, int N);
```

which examines any supplied array, of length N, and returns true if the array is sorted in ascending order, and returns false if it isn't.

Test your function for a range of arrays of different length, some sorted and some unsorted.

(2) Sorting an array using Bubble Sort

In test1.cpp, write a new function

```
void bubbleSort(int* a, int N);
```

which will sort the supplied array of size N into ascending order, using the Bubble Sort algorithm discussed in lectures.

Test your method first on a small known array, and then on an array of random numbers. Validate your function using the isSorted function above. Then time the operation for several values of N (10000, 20000, …, 50000) and record the times.

(3) Sorting an array using Quick Sort

In test1.cpp write a new function

```
void quickSort(int* a, int first, int last);
```

which sorts the supplied array, from the first element to the last element, into ascending order, using the Quick Sort algorithm discussed in lectures. Your function should return the result in the original array a. Again, test your method on an array of random numbers, and validate your function using isSorted. (Note: don't run it on the already-sorted result of bubbleSort! You can back up the array a using an array b, before a is sorted by bubbleSort, and then sort b using quickSort for comparing the times between the two algorithms).

To check that you understand the algorithm, print out the array inside the function after the partitioning phase, but before the sorting phase (i.e. before the recursive calls to quickSort). Also print out the value of ‘i’ (the start position of the upper half after partitioning). Check carefully that you understand exactly why each value ended up exactly where it is.

For example, do this exercise on paper. Trace the program from the code (don’t run it):

int b[] = {41, 8, 33, 16, 14, 56, 50, 5};

After 1st partitioning (don’t run the program on this array yet! Do it by hand):

b will be: …………………………………… i (start of right ‘half’) will be ………….

When sorting the left ‘half’: after partitioning the left half:

it will be: …………………………………… Its i will be ………….

When sorting the right ‘half’ of the original partitioned b: after partitioning the right half:

it will be: …………………………………… Its i will be ………….

Now, run your program to see if your answers are correct. It should output the values of the array segments after each partitioning phase. See if you made any mistakes in your predictions – and learn from them!

Finally, time the operation for N = 10000, 20000, …, 50000 and record the times.

If you have any questions, ask a demonstrator or a teaching staff face to face in the labs, or online by using MS Teams, during the timetabled practical session time. At any other time outside the timetabled session time, you can send your questions by using the Ticketing System, or by using emails.

Practical Test 2

Practical Test 2 will take place on Wednesday 15th December. The test will mainly cover the taught material from Lecture 6 to 9, and practical material from Practical 6 to 9. Details of the test arrangement will be made available on CSC2059 Canvas.