Assignment 2
Algorithms and Data Structures 1 (1DL210)
2021

Erik Edlund Simon Ström

October 6, 2021

Instructions

- Unless you have asked for and received special permission, solutions must be prepared in groups. If you got special permission for the first assignment, you also have special permission for this assignment.
- Make sure that your programs run on the UNIX/LINUX system. Any (sub)solutions that don't run on the UNIX/LINUX system **will** receive 0 points. In particular, you should verify the output of your implementation against the output of the UNIX/LINUX sorting command line `sort -n`.
- When you have completed the assignment, you should have four files: three sorting programs and one Pdf document explaining the differences between them. Make a `.zip` file containing these four files.
- Submit the resulting `.zip` file to Studium. Only **one** of you from each group needs to do this.

Implementing Sorting Algorithms

For this assignment, you are going to implement three different sorting algorithms based on their textbook descriptions that you can find in the lecture slides.

- Each algorithm is to be implemented in separate files.
- The file sort.py contains skeleton code to run your algorithm. Call your sorting algorithm inside respective functions of sort.py. For example, bubblesort is implemented and called at line 32 inside `run_bubblesort()` function.
- The program should read a file called `nums.txt`, which can be assumed to contain integers separated by newlines.
- `rangen.py`(or `rangen_py3x.py` for python 3x) takes an integer argument n and outputs a file `nums.txt` containing n rows with random numbers in the range $\{0, \ldots, n\}$. Example of usage (on python **3.x**): `python rangen_py3x.py 100`
- Output of each algorithm is stored in a file. For example, for insertion sort the file is `insertionsorted.txt`.

- Additionally, you should verify that your result matches the output of the `sort -n` Unix/Linux system command. Running`test.sh` will do for you.
- If you are not implementing your algorithm in python or not using the python skeleton code, make sure that your program behaves similarly. In this case, you probably need to re-implement yourself a `run()` function to run your algorithms.
- Your code should produce output files with the names `insertionsorted.txt`, `quicksorted.txt`, and `heapsorted.txt`. Each of these files should contain output sequence as integers separated by newlines.

Note that the arrays in the lecture and in the textbook are indexed from 1, whereas in most programming languages they are indexed from 0.

Insertion Sort (3p)

Implement Insertion-Sort from the second lecture.

Quicksort (3p)

Start by implementing Partition. Then use it to implement Quicksort from the fourth lecture.

Heapsort (3p)

Implement the functions Max-Heapify, Build-Max-Heap, and Heapsort from the fifth lecture.

Comparison (1p)

What are the differences between the three algorithms? For each algorithm, mention a situation where it has an advantage over the other two.

Answer:

The Insertion sort algorithm splits an array in two, and then focuses on one half to make it sorted, by swapping its elements places after their values. The algotihm compares the value of each element with the value of the element above, and everytime a decreasing order of values is found - the elements are swapped. The lower found element are then compared with every element to the left of its new place, until its left element has a lower value than itself.

In Quick sort a pivot element are chosen. It's then put at the end of the array. Then the array are scanned from the beginning, until a larger element than the pivot are found. The same is done from the opposite direction right the element just left of the pivot, but instead in search for a smaller element than the pivot. When this pair are found, those the elements are swapped. The process is repeated until no such pairs in the wrong order can be found. The pivot are then put back at the place of the latest found larger element.

The heap sort algorithm uses binary trees where max heap parents are put at the top, with lower so called childs underneath.

Insertion sort is the fastest algorihtm for short arrays. For those, the quick sort algorithm would be unefficient and would require unnecessary much memory.

The quick sort are however often the fastest alternative for sorting an array with unsorted elements. That's because unnecessary swaps not are done, since the array are scanned after swaps that are necessary to be done. However, if the array are not far from being sorted, the insertion sort could be a faster option.

Heap sort is both a fast and accurate choice. Heap sort is a good choice considering the memory, since it doesn't require any extra data structure. Its time complexity in the worst case is $\mathcal{O}$(n log n).