Algorithms[*] Lab 4

1 Review

Topics covered this week:

- More sorting: heapsort, quicksort, sorting lower bound

In lab exercises: COLLABORATION LEVEL : 0

1. What is the running time of Quicksort when all elements of the array have the same value?
2. Briefly sketch why the running time of Quicksort is $\Theta(n^2)$ when the array contains distinct elements and is sorted in decreasing order.
3. Lomuto partition: Let $A = \{3, 6, 1, 5, 8, 2, 4, 1, 3\}$, and assume we call Quicksort$(A, 0, 8)$. Show what happens during the first invocation of Lomuto Partition. What is the value of q returned, and what are the two recursive calls made?
4. Lomuto and Hoare partition: In class we went over LOMUTO partition. Sir Hoare, when describing Quicksort, proposed a different partition referred to as the HOARE PARTITION. You can find it in the textbook (problem 7-1, page 185),or, you can google it.

   Let $A = \{3, 6, 1, 5, 8, 2, 4, 1, 3\}$, and assume we call Quicksort$(A, 0, 8)$. Show what happens during the first invocation of Lomuto Partition. What is the value of q returned, and what are the two recursive calls made?

   Is it stable? How does it handle elements that are equal to the pivot? What happens when it is run on a sequence of equal elements? How would you compare Lomuto Partition with Hoare Partition?
5. A sorting algorithm is called *stable* if it leaves equal elements in the same order that it found them in.

   Go through the sorting algorithms that we learnt, and analyse if they are stable (or can be easily made stable) or not. Specifically: Is Heapsort stable? Argue, or give a counterexample.
6. (C-4.22) Let A and B be two sequences of n integers each. Given an integer x, describe an $O(n \lg n)$ algorithm for determining if there is an integer a in A and an integer b in B such that $x = a + b$.

[*]csci2200, Laura Toma, Bowdoin College

Homework:

Pair-programming honor code: For this lab you will work with one partner and do pair-programming. The honor code for pair-programming is the following:

1. You must work together, physically in the same place.
2. The overall lab must be a true joint effort, equally owned, created and understood by both partners.
3. Specifically splitting the lab into parts and working on them separately is not allowed and violates the honor code for the class.
4. If you start together as a team and are unable to complete the project together, then you will each inherit the shared code, and split up to work individually on the remainder of the project. You would then submit individually, with a note on what happened.
5. The two partners in a team obviously share everything, obviously. Across teams, collaboration is allowed at COLLABORATION LEVEL: 1.

Use a language of your choice in $\{Java, Python, C, C++\}$ to implement, test and evaluate Quicksort and Heapsort.

- If you follow the pseudocode from class notes or textbook, you will use plain C. Your source files should have the `.c` or `.cpp` extension. Don't worry about a header file. If you write C++ code, don't worry about making a class. If you would rather use Python or Java, that's fine.

  The guidelines below are meant for C/C++, if you are using Python/Java, adjust them accordingly.
- Your array should contain random floating point numbers. Write a function that initializes the array with random floats.
- The size of the array can be specified on the command line (ideal), or via a constant at the top of the code.
- Run each one of your sorts in inputs of sizes $N = 1M, 10M, 100M, 1B$ and record the running times.
- You may get errors when allocating such large amounts of memory. Its' worth experimenting with static vs dynamic allocation. Different compilers.languages may have different limits, so expect different results with Python vs Java vs C/C++.
- Your team should have two files: `heapsort.c` and `quicksort.c`. To make grading easier, please prefix the files with the last names in your team. For example, if Duncan and Eric were working together, they would submit two files: `EricDuncan-quicksort.c` and `EricDuncan-heapsort.c`.

- Each file should contain the sort function, a main function that calls the sort function in the appropriate way, and a function that tests that the sorting was correct, like so:

```
/*

File: quicksort.c

Authors: Eric and Duncan
Date:

*/

const int N = 100;

/*
quicksort in place array a of length n;
assume array is allocated and populated with data prior to this call
*/
void quicksort(int* a, int n) {

...

}


/*
return 1 if a is sorted (in non-decreasing order); and 0 otherwise
assume array is allocated and populated with data prior to this call
*/
int issorted(int* a, int n) {
...
}

int main() {

   //declare and allocate the array
   //populate the array with random data
   //call quicksort

   //test that array is sorted
   if (issorted(a,n)) {
         printf("Array is sorted!\n");
   } else {
        printf("OOPS! Array is NOT sorted!\n");
   }
}
```

- To compile from the command line, you can use

```
[ltoma@lobster ~]$g++ EricDuncan_quicksort.c -o EricDuncan_quicksort
```

- Time and print the time it takes to sort. This should not include the time to populate the input with random floats, or the time to test that it's sorted.

What to turn in:

1. **Use piazza to email the code:** Use piazza toprivately email the .c or .cpp or the Python file. Please do not email me any Xcode/NetBeans/ projects —- only the source file. Note: use Piazza, do not use my bowdoin email. Note: send a private email. Note: do not send it anonymously, because I need to know who you are.

2. Bring hard copies of your code, one per team.

3. A brief report containing the table with the running times, along with any thing you learnt while experimenting with large N. Hard copy, stapled together with the hard copy of your code.