

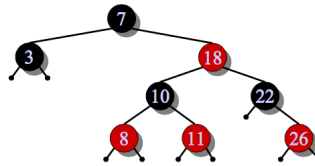
# Algorithms Lab 4

## In lab

1. Red-Black trees: Insert the following values in the RB tree below in this order:

*INSERT*(15), *INSERT*(16), *INSERT*(17)

Show the tree that results after each insertion.



## Homework

Work with one partner. Implement Quicksort and Heapsort. For testing, generate random floating point numbers. Write your code so that the sorting is timed separately from initialization. Test on inputs of size  $n = 1000, 1M, 10M, 100M, 1B$ , and record the timings in a table.

- 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. This is a chance to review C/C++; if you would rather use Python, that's fine.

The guidelines below are meant for C/C++, if you are using Python, adjust them accordingly.

- 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 Ducan and Eric were working together, they would submit two files: `EricDuncan-quicksort.c` and `EricDuncan-heapsort.c`.
- To submit, email me the two files (only once per team, please).
- 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
#ifdef NDEBUG
    if (issorted(a,n)) {
        printf("Array is sorted!\n");
    } else {
        printf("OOPS! Array is NOT sorted!\n");
    }
#endif
}

```

- To compile from the command line, you can use

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

- To time the sort, you can call the timer in your code; or simpler, you can use the `time` command in the shell:

```
[ltoma@lobster ~]$ time EricDuncan_quicksort
```

This will return the total time for the command (real time), and CPU time (user + system), something like this:

```

real 0m0.003s
user 0m0.001s
sys 0m0.002s

```

What to turn in:

- email me the `.c` or `.cpp` or the Python file. Please do not email me any Xcode/NetBeans/projects — only the source file.
- Bring hard copies of your code, one per team.
- A brief report containing the table with the running times. Hard copy, stapled together with the hard copy of your code.