Test Exam

Algorithms and Data Structures

Jacob Trier Frederiksen & Anders Kalhauge

Spring 2019

This Test Exam is on Tuesday, April 23rd, 08:30-12:00.

Be there on time.

NB: We wish to emphasize that this Exam is for your own benefit, so that you may get a real view of where you stand w.r.t. readiness for the upcoming final exam. You will not be graded, but merely have an indication of whether you are at a 'pass' or 'fail' level. Do youselves a favor, and complete this test, on your own, individually.

This Test Exam is not mandatory, but will serve the purpose of an self-evaluation tool.

You can use your computer with tools, and the internet for reference material. However, you should *not* communicate with others. Thus, you should refrain from using services as Facebook, Messenger, Skype, etc. Please close all communication services during the test. Talking with your classmates is disallowed during the test.

Your answers shall be handed in on paper. Please label any scratch paper with calculations/text/drawing clearly, if you need more space than the test sheets alone.

Full name: Jacob Trier Frederiksen

Number of additional pages (apart from the test exam sheets) 1

$\mathcal{O}$

1 Determine the $\mathcal{O}$ time complexity with respect to n of the following methods

Use the standard $\mathcal{O}()$ – or "Big O" – notation in your answer. Remember that $\mathcal{O}(n-1)$ is the same as $\mathcal{O}(n)$, fx., when we determine the scaling of an algorithm.

Hint: check loop dependencies, and in particular dependencies on n here...

1.1

```
double q1(int n) {
  double r = 0.0;
  for (int i = 0; i < 10000; i++) {
    r = Math.sqrt((double)n/i);
    }
  return r;
  }
```

$\mathcal{O}(1)$

loop does not depend on 'n'. Always const. loop size. Sqrt does not depend on 'n'.

1.2

```
double q2(int n) {
  double r = 0.0;
  for (int i = 1; i < n; i++) {
    r += Math.log(i);
    }
  return r;
  }
```

$\mathcal{O}(n-1) \equiv \mathcal{O}(n)$

n-1 iterations

1.3

```
double q3(int n) {
  double r = 0.0;
  while (n > 1) {
    r += Math.log(n);
    n = n/2;
    }
  return r;
  }
```

$\mathcal{O}(\log n)$

Problem is halved in every iteration.

1.4

```
static double q4(int n) {
  int r = 0;
  for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
      r++;
      }
    }
  return r;
  }
```

$\mathcal{O}(\frac{n^2}{2}) = \mathcal{O}(n^2)$

$n \cdot \frac{n}{2} \simeq \frac{n^2}{2} \approx n^2$

2 Improving $\mathcal{O}$ of a slow algorithm

The "ThreeSum" algorithm from the "Booksite" (i.e. 'Algorithms, 4th Edition', Sedgewick/Wayne, Princeton), has a particular scaling in terms of $\mathcal{O}$, for n (compare with the code snippet in 1.4 above for q4). – What is that scaling?

Scaling is $\mathcal{O}(n^3)$, two nested loops.

There is a faster version of ThreeSum, which can obtain a significantly better scaling (in the variable n, again). "ThreeSumFast"

- What is the name, and scaling of the faster version of ThreeSum? Scaling: $N^2 \log N$
- What are the modifications to the slow ThreeSum that makes this improvement possible? Add sort, add binary search ⇒ only one nested loop.

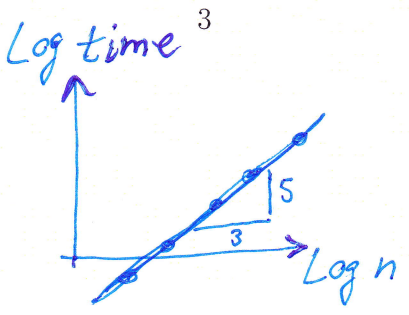
3 Experimental scaling, $\mathcal{O}$ time complexity

For a computing time scaling experiment, an algorithm (unnown to us), exhibited a scaling with respect to n for an algorithm with the running time as stated below.

n	time
125	0,03s
1.000	1,00s
8.000	32,00s
64.000	1.024,00s
512.000	32.768,00s

⇒ Do what is stated in the hint.

Hint: remember our Excel plotting exercise from a few weeks ago. You can find help in the course repo (Week14, excel file), or in the book (ch.1.4). Plot the numbers in a scatterplot, and with Log-Log-axes. Then you can find the slope (in Log-Log coordinates) and thus arrive at the answer.



$\frac{5}{3} \Rightarrow \mathcal{O}(n^{\frac{5}{3}})$.

4 State the order of growth

Write the running time complexity, $\mathcal{O}$ dependence on n, for the following algorithms in the special case of sorting a list initially given in reverse order, e.g. $(99, 56, 45, 32, 13, 9, 7)$:

4.1 Quick Sort $O(n^2)$, worst case scenario.

4.2 Insertion Sort $O(n^2)$

4.3 Heap Sort $O(n \log n)$

4.4 Selection Sort $O(n^2)$

5 Pros and Cons, sorting complexity

Write down as many pros and cons (time, space, stability, ...) for the sorting algorithms:

- Quick Sort pros: low overhead memory, NlogN (but worst is N^2, although unlikely) cons: unstable
- Merge Sort pros: stable, always NLogN cons: $O(N)$ memory overhead
- Heap Sort pros: in-place, guaranteed NlogN, great for prio-queues. cons: unstable

Hint: you may find it useful to list some properties in a table, and comment...

Sorting

6 Match the sorting algorithms

The following array of numbers is unsorted:

47	14	39	26	31	42	12	55	17	33

.

Determine the sorting algorithms, which partially sorted the lists below, for each case. In other words; which algorithm has started sorting, but is not yet finished?

NB: Green cells are sorted, red cells are moved, black cells are untouched.

6.1

47	14	39	26	31	42	12	55	17	33
14	47	39	26	31	42	12	55	17	33
14	39	47	26	31	42	12	55	17	33

Insertion Sort

6.2

12	14	39	26	31	42	47	55	17	33
12	14	39	26	31	42	47	55	17	33
12	14	17	26	31	42	47	55	39	33

Selection sort.

6.3

14	47	26	39	31	42	12	55	17	33
14	26	39	47	12	31	42	55	17	33
12	14	26	31	39	42	47	55	17	33

Merge-sort (bottom-up).

Hint: candidate sorting algorithms for the above partially sorted arrays are Quick Sort, Insertion Sort, Merge Sort, Heap Sort, and Selection Sort.

Searching

7 Searching

What are the scaling differences ($\mathcal{O}$) between binary search in an ordered array and search in a binary tree?

None. Both are $\mathcal{O}(\log n)$, average case.

8 A Binary Search Tree

Draw a binary search tree of the letters in your full name (omit white spaces).

- Is your tree "reasonbly" balanced?
- How can we quantify the balance of a binary tree, in terms of the structure size scaling with n?

See Appendix A

Hint: think deep ...

9 Find the maximum

```
class BinaryTree<T extends Comparable> {
  private T data;
  private BinaryTree<T> smaller;
  private BinaryTree<T> larger;
  ...
  public T maximum() {

    // Maximum code goes here... it's a two-liner...
    //
    // Hint:
    //          if ( ? == ?? ) ret...
    //          retu...;

    }

  }
```

if(larger == null) return data;
return larger.maximum();

Fill in the missing code in the `maximum` method.

10 Balanced Search Trees (BSTs)

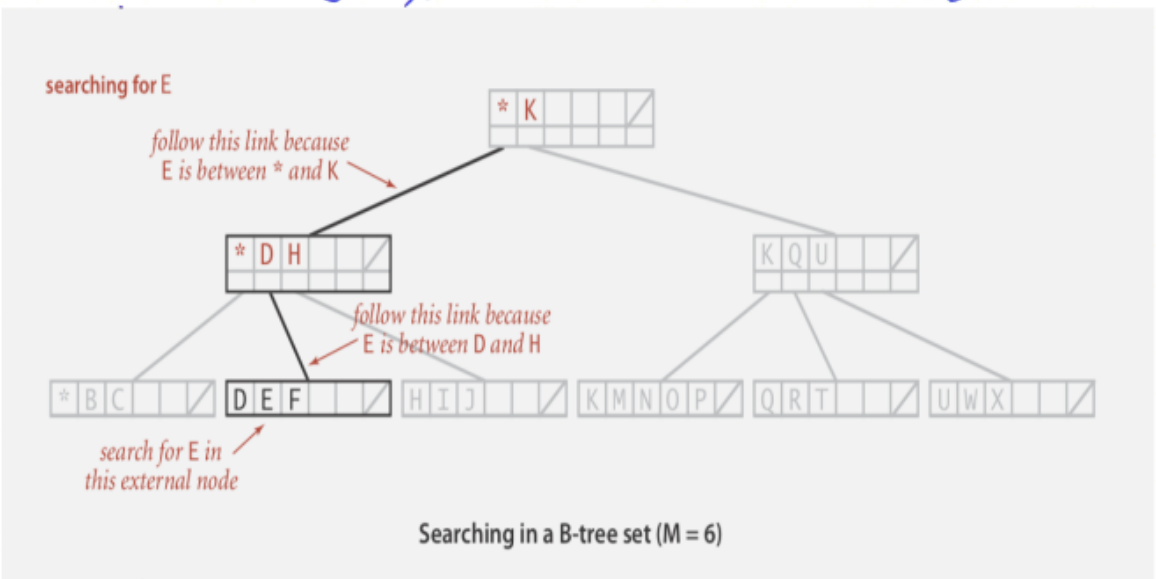
Name an example of a BST data structure. What are some of the advantages – and disadvantages – compared to classic Binary Search Tree?

Three examples: "2-3 Search Trees"

"Red-Black BSTs"

"B-Trees"

The B-Tree: "perfect balance" due to multiple key-link pairs.
in practice guaranteed about 4-5 probes for insertion and search.
B-Tree, order M - with N keys ⇒ $O(\log_{\frac{M}{2}} N) \sim 4$ or 5.

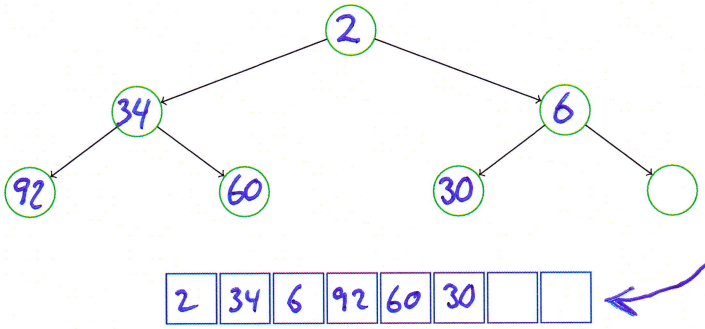


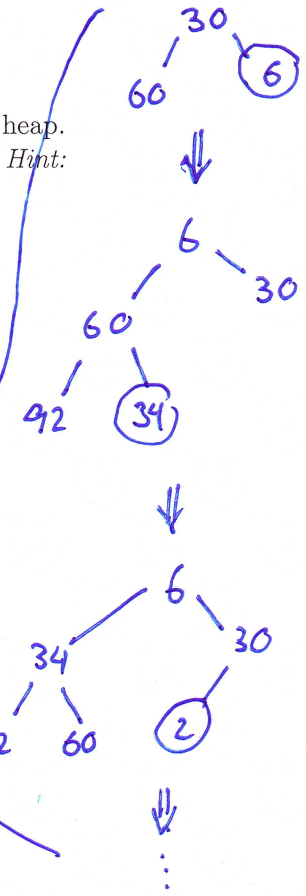
Searching in a B-tree set (M = 6)

Queueing

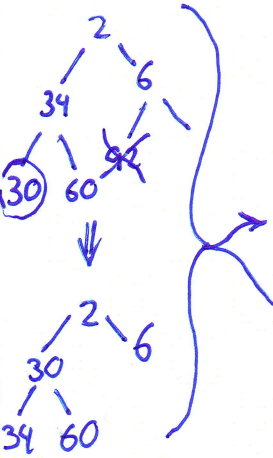
11 Heaps

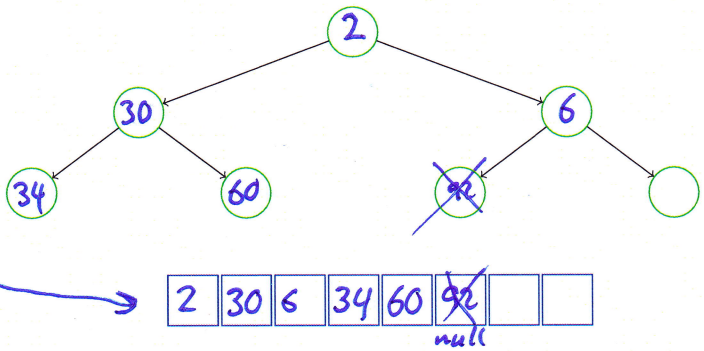
Structure the following input $(30, 60, 6, 92, 34, 2)$ in a minimum oriented heap. Draw the steps in heapifying the data, using "sink" and "swim" operations. *Hint: you will need a bit more than 4 steps.*



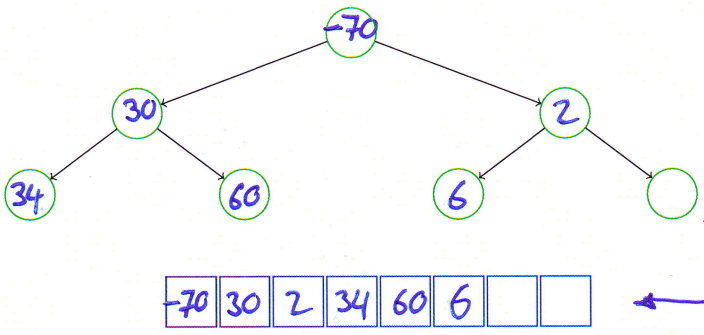


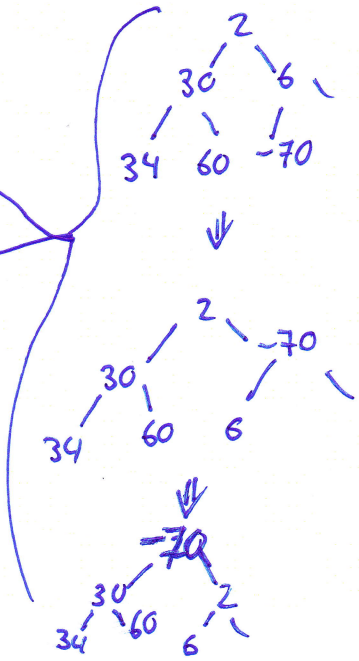
Delete 92 from heap. *Hint: you will need less than 4 steps.*





Add -70 to the heap. *Hint: you will need between 2 and 4 steps.*





Graphs

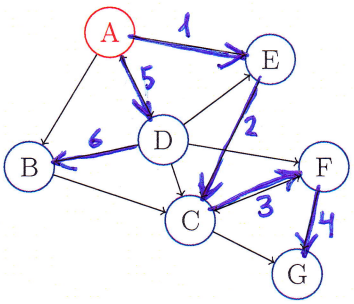
12 Graphs

Below, exhaustively list as many concepts and terms as you can remember from the cirruculum on graphs. *Hint: things like 'un-directed graph', 'vertex', $\mathcal{O}$, data structures, shortest paths, etc.....*

There are many! ☺.

13 Iterative Depth-first traversal

Show the iterative depth-first traversal from node 'A' on the graph below:

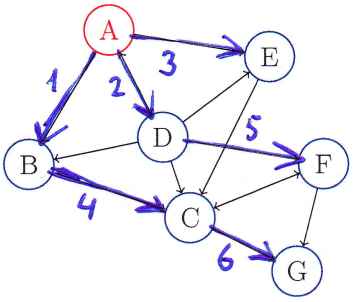


1	A-B	A-D	A-E	
2	A-B	A-D	E-C	
3	A-B	A-D	C-G	C-F
4	A-B	A-D	C-G	F-G
5	A-B	A-D		
6	~~A-B~~	D-B		

Write down the list of changes to the stack, and sequence (number the edges) of traversal, to support your solution.

14 Iterative Breadth-first traversal

Show the iterative breadth-first traversal from node 'A' on the graph below:

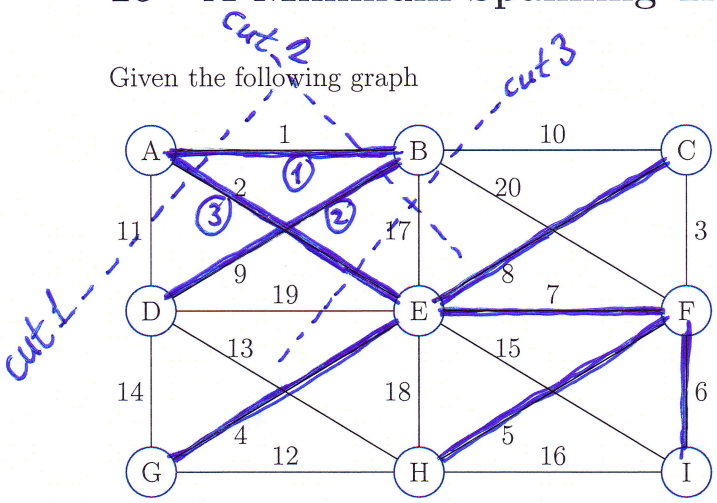


1	A-B	A-D	A-E	
2	A-D	A-E	B-C	
3	A-E	B-C	D-C	D-F
4	B-C	~~D-C~~	D-F	E-C
5	D-F	~~E-C~~	C-G	C-F
6	C-G	~~C-F~~		

Write down the list of changes in the queue, and sequence (number the edges) of traversal, to support your solution.

15 A Minimum Spanning Tree (MST)

Given the following graph

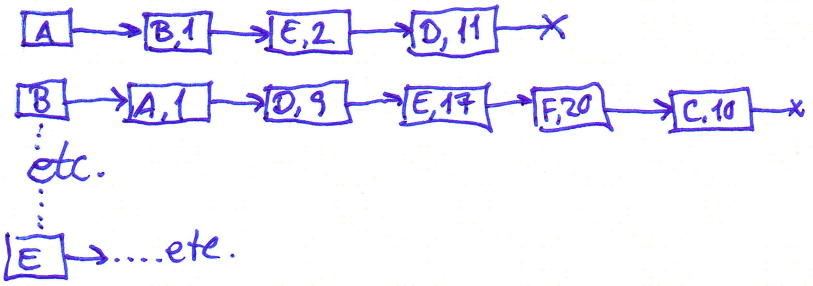


- Draw on the graph, the minimum spanning tree. Give an explanation about the algorithm used. Used the greedy brute force cut algo.
- Write edges and vertices as sets

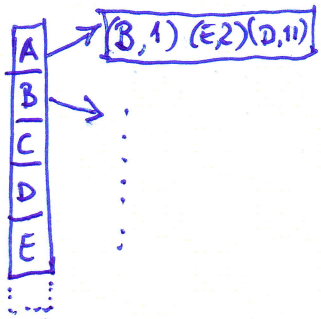
$V = \{A, B, C, \ldots, H, I\}$

$E = \{(A,B,1), (B,C,10), \ldots, (H,I,16)\}$

- Sketch, partially, an edge list data structure for the first - say - 5 nodes, i.e. A to E.



- Sketch, partially, an array of adjacency lists data structure, for the first - say - 5 nodes, i.e. A to E. (same-same, but different).



- What would be the time complexity (big O) of finding if two vertices are connected?

O(1) : look-up of vertex ⇒ List of adjacent Vs.

[O(n) for checking if two vertices are connected.]

16 An adjacency matrix

Below, draw the graph as an adjacency matrix.

- What would be the time complexity (big O) of finding adjacent nodes?

  $O(1)$: just a look-up.

- What would be the time complexity (big O) of finding if two nodes are adjacent?

  $O(n)$: check all elements i row for given vertex.

- What would is the memory consumption scaling (big O) with V, the number of vertices?

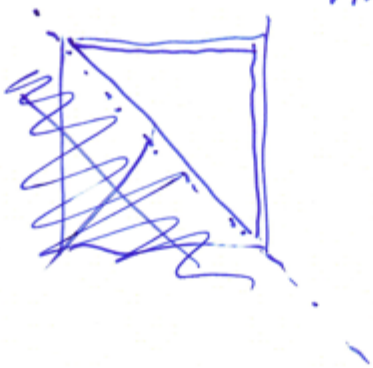
  ~~$O(1)$: just look up~~. $O(V^2)$: quadratic since 2D-matrix.

- Can you optimize the data structure, i.e. reduce the memory consumption?
  Hint: look for matrix symmetries..., think folding paper...

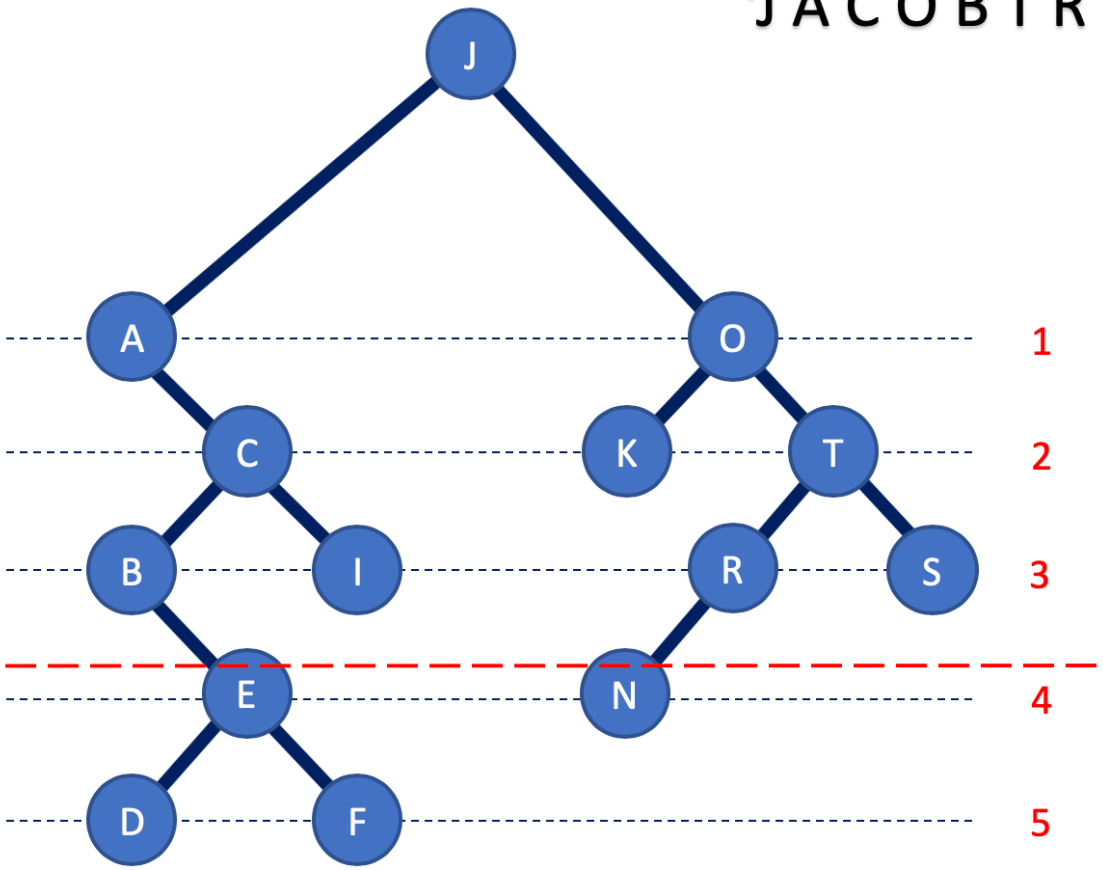
All info in upper triangle.

$\Rightarrow V^2 \xrightarrow{\text{OPT}} \frac{V^2}{2}$

(but it is still $O(V^2)$!).



‘J A C O B T R I E R F R E D E R I K S E N’



Depth is 5.

Close to $Log_2(14)=3.8$, where 14 is the number of unique letters in name string, ignoring blanks. Therefore reasonably balanced binary search tree....

R R E E R I E ”Leftovers”

Appendix A