University of Zurich[UZH]

Department of Informatics
Database Technology Group
Prof. Dr. M. Böhlen

Informatics II
Exercise 5

March 16, 2020

Heap and Heapsort

Task 1. The `d-ary` heap is a generalization of the binary heap in which the nodes have **d** children instead of 2. Write a C program that sorts an array of integers in *ascending* order using `d-ary` heap sort and that prints the sorted array. Your program should first convert the given array into a max `d-ary` heap and print it in the standard output using the requested format.

Your program should include the following:

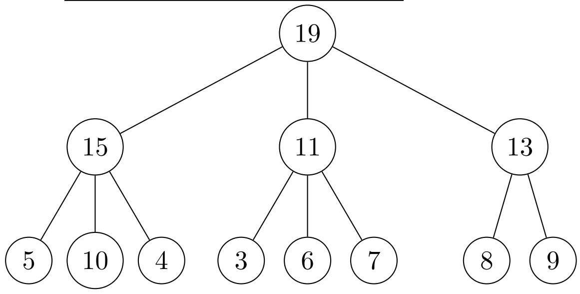
1. The function *void buildMaxHeap(int A[], int n, int d)*, where A is the array to be converted, n is the real size of the array A and d is the maximum number of child each node can have in the heap.
2. The function *void printHeap(int A[], int n)* that prints the created max-heap to the console in the format **`graph g`** { (all the edges in the form `NodeA -- NodeB`) }, where each edge should be printed in a separate line. The ordering of the edges is not relevant.
3. The function *void heapSort(int A[], int n, int d)* that sorts the array A in ascending order.
4. The function *void printArray(int A[], int n)* that prints a given array to the console.

For example, given the array `A = [6, 4, 3, 9, 5, 10, 15, 19, 11, 7, 8, 13]` your program should produce the max-heap `[19, 15, 11, 13, 5, 10, 4, 3, 6, 7, 8, 9]` and print it to the console in the form illustrated on the left figure. Print the content of array A after calling *heapSort* on it. The alternative output is used to check the correctness of your implementation, it does not have to be part of your solution.

Output Form

```
graph g {
  19 -- 15
  19 -- 11
  19 -- 13
  15 -- 5
  15 -- 10
  15 -- 4
  11 -- 3
  11 -- 6
  11 -- 7
  13 -- 8
  13 -- 9
}
```

Alternative Output Form



Test your program with the array `[4, 3, 2, 5, 6, 7, 8, 9, 12, 1]`.

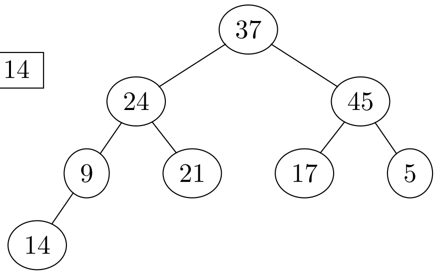
Note: You can copy the output of your program and use it on `http://www.webgraphviz.com/` to view your min-heap in the form of a binary tree, as illustrated on the right figure.

Task 2. Given the array [37, 24, 45, 9, 21, 17, 5, 14], sort it in ascending order using Heapsort algorithm. Precisely explain every step performed. You can start from making max-heap from initialy binary tree shown below. Show every node exchange that occur.

Input array

37	24	45	9	21	17	5	14

Initial binary tree



Quicksort

Task 3. In a $dual - pivot$ quicksort implementation, an array $A[low...high]$ is sorted based on two pivots $p_1 = A[low]$ and $p_2 = A[high]$. The pivot p_1 must be less than p_2, otherwise they have to be swapped. Here low is the left end of the array A and $high$ is the right end of the array A.

The partitioning process is the core of the $dual - pivot$ quicksort. We begin partitioning the array in three proper partition I, II or III, where:

- **Partition I:** In first partition all elements will be less than the left pivot p_1.

- **Partition II:** In the second partition all elements will be greater or equal to the left pivot p_1 and also will be less than or equal to the right pivot p_2.
- **Partition III:** In first partition all elements greater than the right pivot p_2.

After the partitioning, the two pivots are placed in their proper and final positions as illustrated in Figure 1 and the process is repeated recursively on each of the three partitions.

I		II		III
$< p_1$	$\mathbf{p_1}$	$>= p_1 \& < p_2$	$\mathbf{p_2}$	$> p_2$

Figure 1: Quicksort with dual-pivot

Create a C program that implements $dual - pivot$ to sort an array A of integers in *increasing* order. Your program should also print the array before and after the sorting. For example, if the array A to be sorted is `[10, 7, 3, 15, 6, 2, 5, 1, 17, 8]`. your program should print the following in the standard output:

```
Before:  [ 10 7 3 15 6 2 5 1 17 8 ]
After:  [ 1 2 3 5 6 7 8 10 15 17 ]
```