Informatics II
Exercise 5 / **Solution**

March 23, 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.

```c
void heapify(int A[], int i, int n, int d) {
  int max = i;
  int k, child;
  for (k = 1; k<=d; k++){
    child = (d*i) + k;
    if (child < n && A[child]> A[max]) { max = child; }
  }
  if (max != i) {
    swap(A, i, max);
    heapify(A, max, n, d);
  }
}

void buildMaxHeap(int A[], int n, int d) {
  int i;
  for (i = (n-1)/d; i >=0; i--) {heapify(A, i, n, d);}
}
```

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.

```
void printHeap(int A[], int n, int d) {
  int i, l, r, k;

  printf("graph g {\n");
  for (i = 0; i < n; i++) {
    for (k = 1; k<=d; k++){
       if ((d*i) + k < n) { printf("  %d -- %d\n", A[i], A[(d*i) + k]); }
    }
  }
  printf("}");
}
```

3. The function *void heapSort(int A[], int n, int d)* that sorts the array A in ascending order.

```
void heapSort(int A[], int n, int d) {
  int i, s = n;

  buildMaxHeap(A, n, d);
  for (i=n-1; i>0; i--) {
    swap(A, i, 0);
    s--;
    heapify(A, 0, s, d);
  }
}
```

4. The function *void printArray(int A[], int n)* that prints a given array to the console.

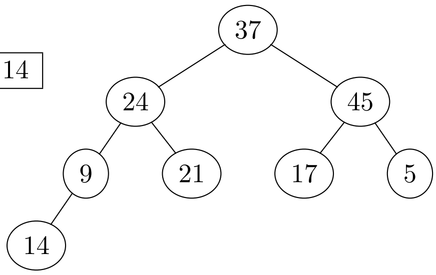
```
void printArray(int A[], int n) {
  int i;
  printf("[ ");
  for (i = 0; i < n; i++) {
    printf("%d", A[i]);
    if (i < n-1) {printf(", ");}
  }
  printf(" ]\n");
}

void printHeap(int A[], int n, int d) {
```

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

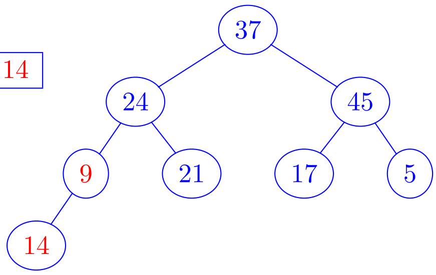


Making max-heap
Step 1

Input array

37	24	45	9	21	17	5	14

Binary tree

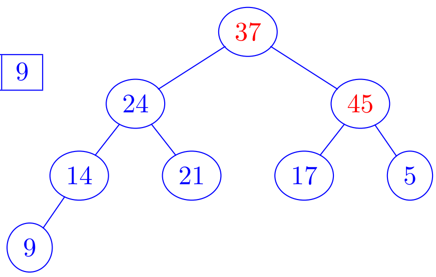


Step 2

Input array

37	24	45	14	21	17	5	9

Binary tree

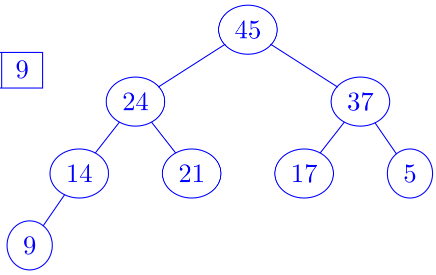


Max-heap

Input array

45	24	37	14	21	17	5	9

Max-heap

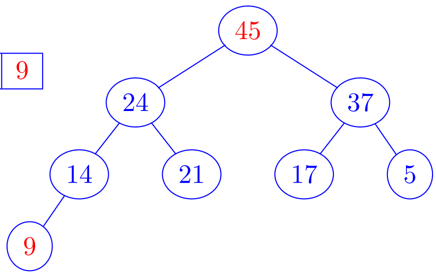


Heapsort
Iteration 1
Exchange last and first(root) element

Input array

45	24	37	14	21	17	5	9

Max-heap

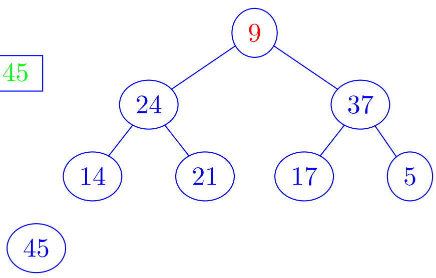


Reorder the tree to make max-heap again

Input array

9	24	37	14	21	17	5	45

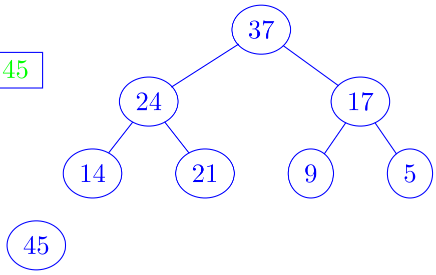
Max-heap



Input array

37	24	17	14	21	9	5	45

Max-heap

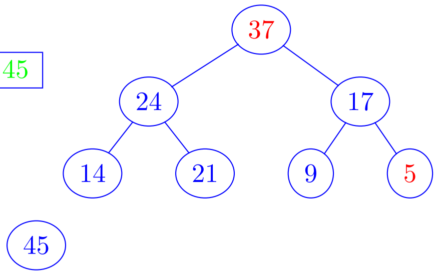


Iteration 2
Exchange last and first(root) element

Input array

37	24	17	14	21	9	5	45

Max-heap

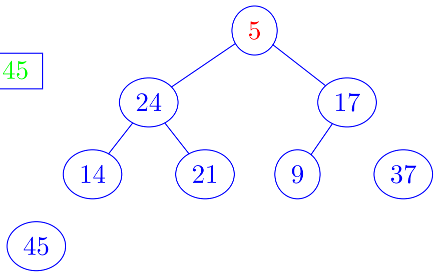


Reorder the tree to make max-heap again

Input array

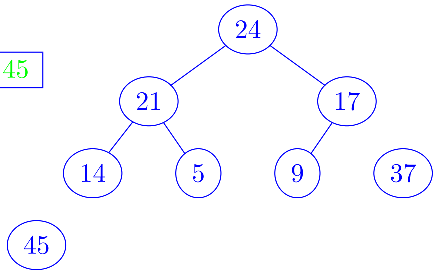
5	24	17	14	21	9	37	45

Max-heap



Input array **Max-heap**

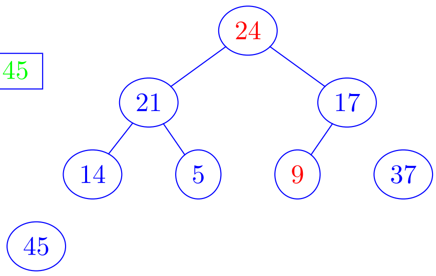
24	21	17	14	5	9	37	45



Iteration 3
Exchange last and first(root) element

Input array **Max-heap**

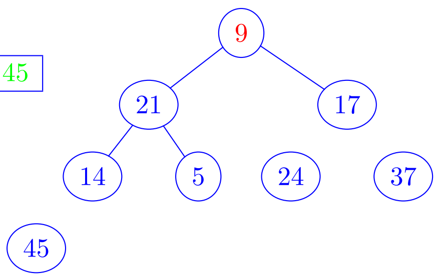
24	21	17	14	5	9	37	45



Reorder the tree to make max-heap again

Input array **Max-heap**

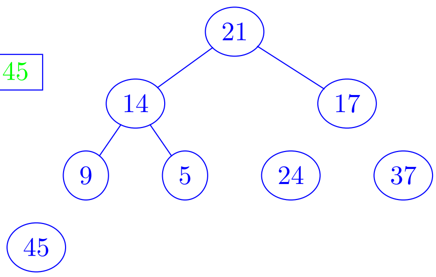
9	21	17	14	5	24	37	45



Input array

21	14	17	9	5	24	37	45

Max-heap

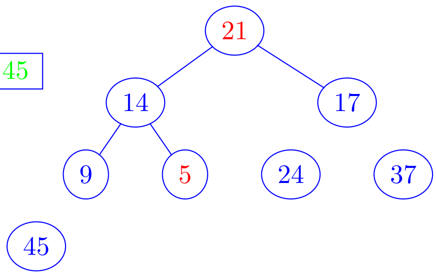


Iteration 4
Exchange last and first(root) element

Input array

21	14	17	9	5	24	37	45

Max-heap

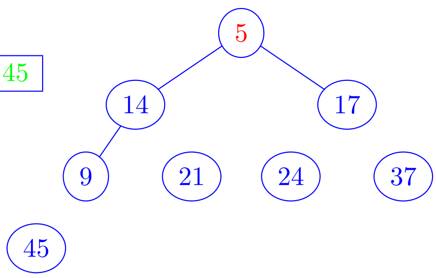


Reorder the tree to make max-heap again

Input array

5	14	17	9	21	24	37	45

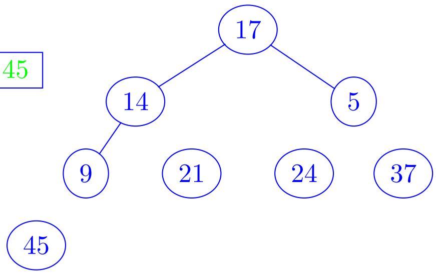
Max-heap



Input array

17	14	5	9	21	24	37	45

Max-heap

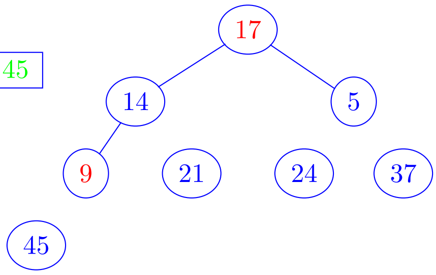


Iteration 5
Exchange last and first(root) element

Input array

17	14	5	9	21	24	37	45

Max-heap

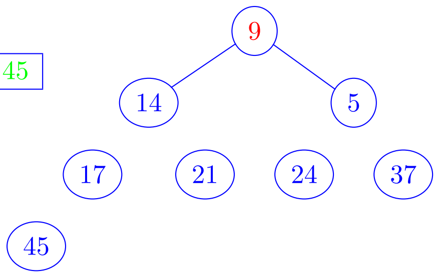


Reorder the tree to make max-heap again

Input array

9	14	5	17	21	24	37	45

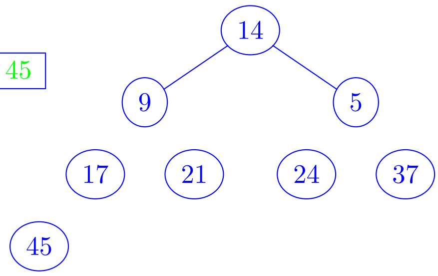
Max-heap



Input array

14	9	5	17	21	24	37	45

Max-heap

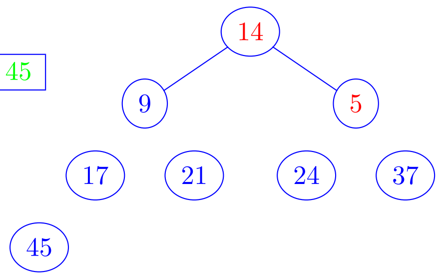


Iteration 6
Exchange last and first(root) element

Input array

14	9	5	17	21	24	37	45

Max-heap

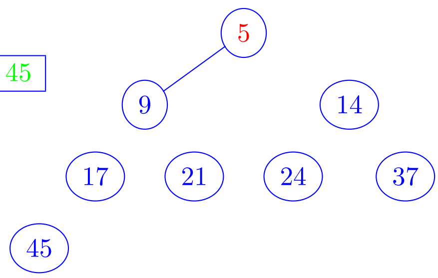


Reorder the tree to make max-heap again

Input array

5	9	14	17	21	24	37	45

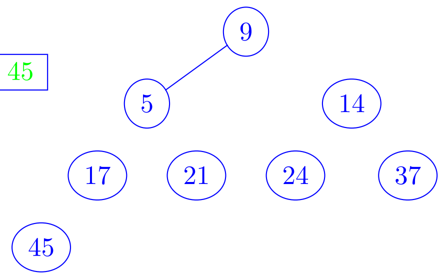
Max-heap



Input array

9	5	14	17	21	24	37	45

Max-heap

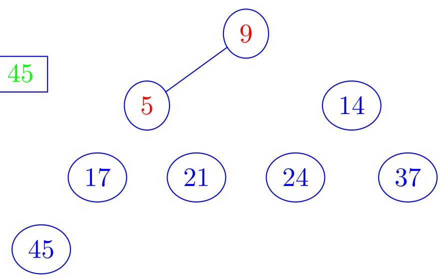


Iteration 7
Exchange last and first(root) element

Input array

9	5	14	17	21	24	37	45

Max-heap

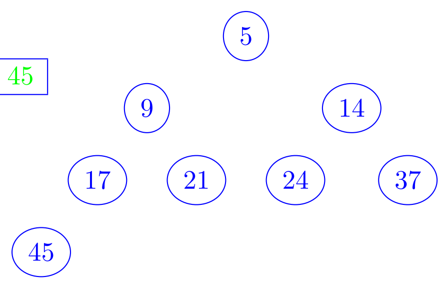


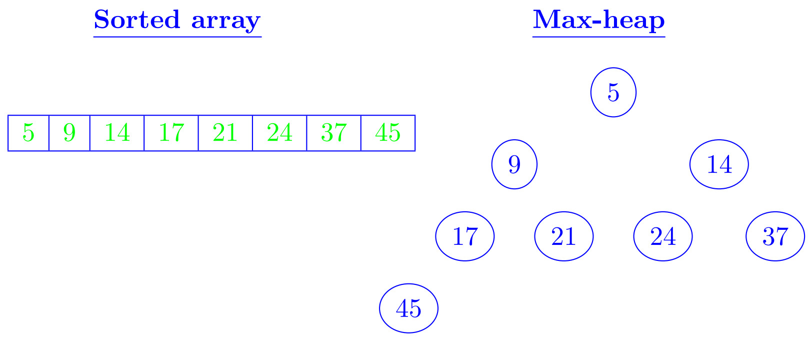
No need for reorder

Input array

5	9	14	17	21	24	37	45

Max-heap





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

Write 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 ]
```

```c
void partitioning(int A[], int low, int high, int *p1, int *p2) {
    int case1 = 0, case2 =0, case3 = 0;
    int l = low+1;
    int k = low+1;
    int g = high;
    while(k < g) {
        case1 = 0; case2 =0; case3 = 0;
        if (A[k] < A[low]) { swap(A, l++, k++); case1 = 1; }
        else if (A[k] >=A[high]) { swap(A, k, --g); case2 = 1;}
        else { k++; case3 = 1;}
        printf("%d - %d - %d === ",case1,case2,case3);
        printArray(A, ARRAY_SIZE);
    }
    swap(A, low, l-1);
    swap(A, high, k);
    *p1 = l-1;
    *p2 = k;
}

void quicksort(int A[], int low, int high) {
    if (high - low <=0) { return; }
    if (A[low] > A[high]) { swap(A, low, high); }
    int p1, p2;
    partitioning(A, low, high, &p1, &p2);
    quicksort(A, low, p1-1 );
    quicksort(A, p1+1, p2-1 );
    quicksort(A, p2+1 , high);
}
```