Informatics II, Spring 2023, Solution 5

Publication of exercise: March 19, 2023
Publication of solution: March 26, 2023
Exercise classes: March 27 - March 31, 2023

Task 1: Heap and Heapsort

1.1
```
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);}
}
```

1.2
```
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("}");
}
```

1.3
```
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);
  }
}
```

1.4
```
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");
}
```

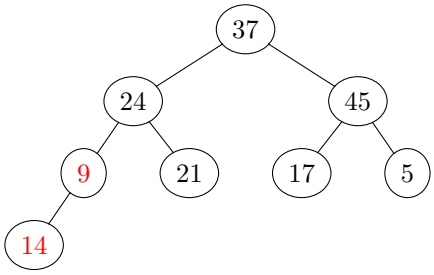
Task 2

Making max-heap
Step 1

Input array

37	24	45	9	21	17	5	14

Binary tree

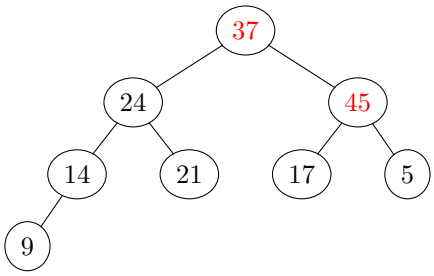


Step 2

Input array

37	24	45	14	21	17	5	9

Binary tree

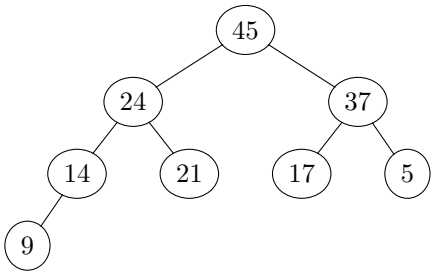


Max-heap

Input array

45	24	37	14	21	17	5	9

Max-heap

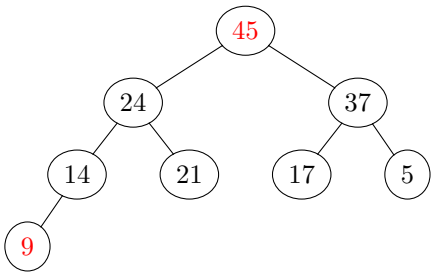


Heapsort
Iteration 1
Exchange last and first(root) element

Input array

45	24	37	14	21	17	5	9

Max-heap

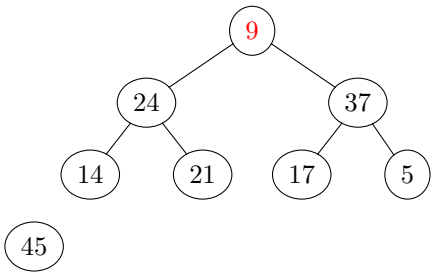


Reorder the tree to make max-heap again

Input array

9	24	37	14	21	17	5	45

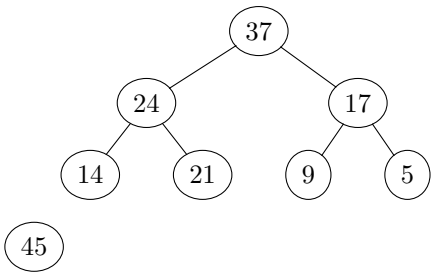
Max-heap



Input array

37	24	17	14	21	9	5	45

Max-heap

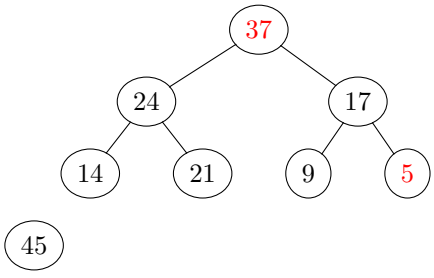


Iteration 2
Exchange last and first(root) element

Input array

37	24	17	14	21	9	5	45

Max-heap

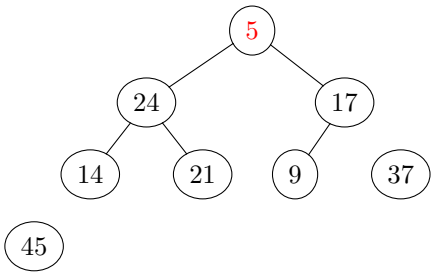


Reorder the tree to make max-heap again

Input array

5	24	17	14	21	9	37	45

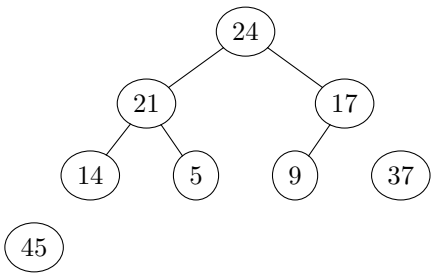
Max-heap



Input array

24	21	17	14	5	9	37	45

Max-heap

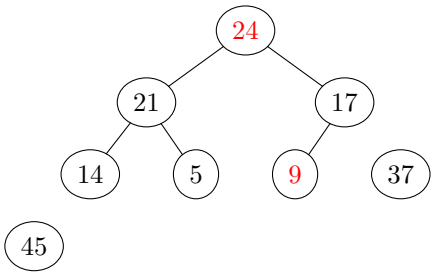


Iteration 3
Exchange last and first(root) element

Input array

24	21	17	14	5	9	37	45

Max-heap

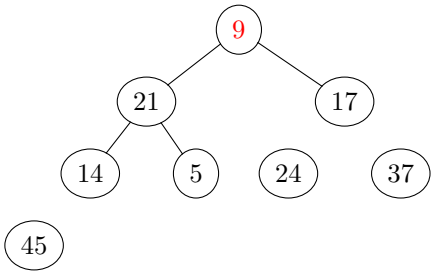


Reorder the tree to make max-heap again

Input array

9	21	17	14	5	24	37	45

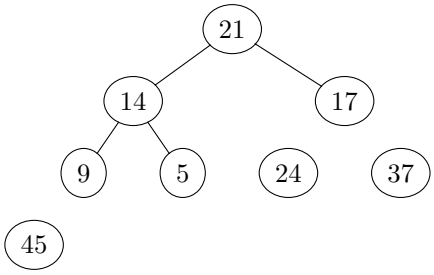
Max-heap



Input array

21	14	17	9	5	24	37	45

Max-heap

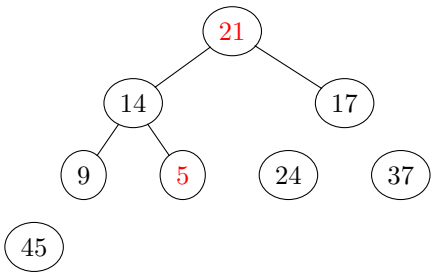


Iteration 4
Exchange last and first(root) element

Input array

21	14	17	9	5	24	37	45

Max-heap

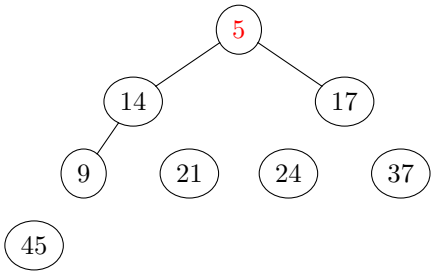


Reorder the tree to make max-heap again

Input array

5	14	17	9	21	24	37	45

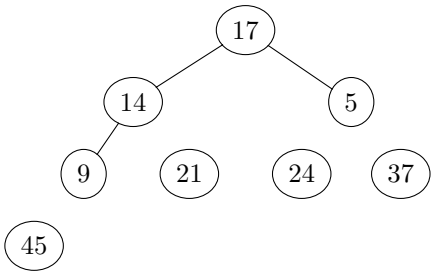
Max-heap



Input array

17	14	5	9	21	24	37	45

Max-heap

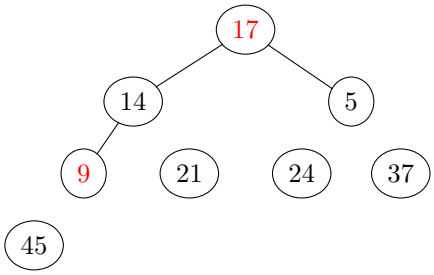


Iteration 5
Exchange last and first(root) element

Input array

17	14	5	9	21	24	37	45

Max-heap

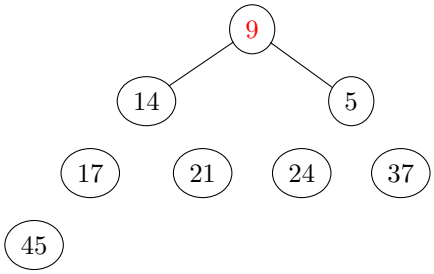


Reorder the tree to make max-heap again

Input array

9	14	5	17	21	24	37	45

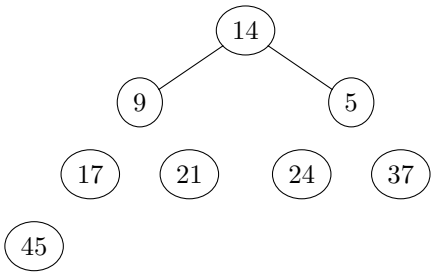
Max-heap



Input array

14	9	5	17	21	24	37	45

Max-heap

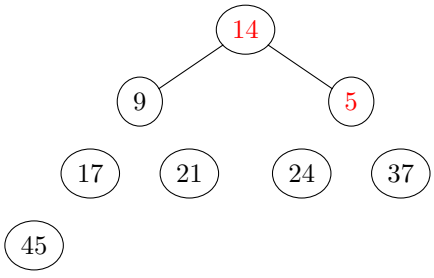


Iteration 6
Exchange last and first(root) element

Input array

14	9	5	17	21	24	37	45

Max-heap

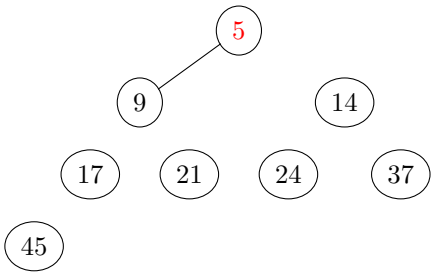


Reorder the tree to make max-heap again

Input array

5	9	14	17	21	24	37	45

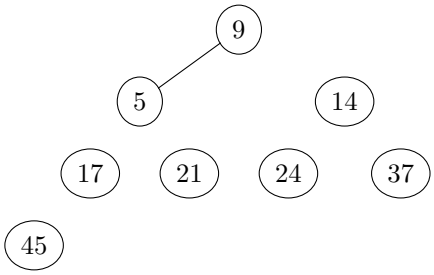
Max-heap



Input array

9	5	14	17	21	24	37	45

Max-heap

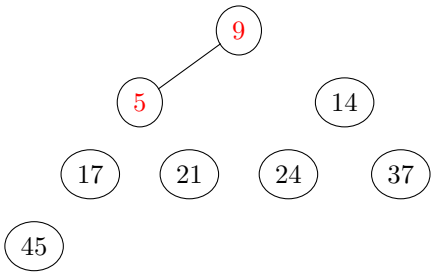


Iteration 7
Exchange last and first(root) element

Input array

9	5	14	17	21	24	37	45

Max-heap

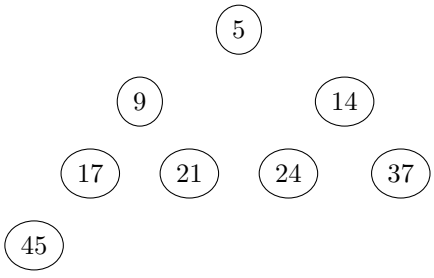


No need for reorder

Input array

5	9	14	17	21	24	37	45

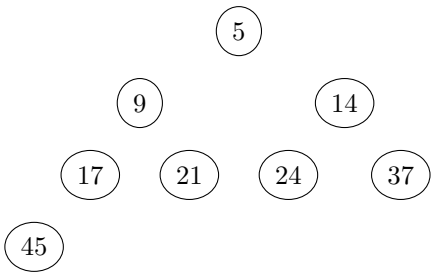
Max-heap



Sorted array

5	9	14	17	21	24	37	45

Max-heap



Task 3: Quicksort

```
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;}
        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);
}
```