Lectures 22, 23, 24

1. Heap Sort
   (i) The correspondence of indices of an array and the complete binary tree obtained from it (textbook page 630).
      i – the array index
      l – the level inside the tree
      p – the position inside the level l, p=0,…,2^l-1
      Take i, the index of an array element and compute l and p from the formula i+1=2^l+p
   (ii) Heap Sort – array indices
      i – the array index
      [(i-1)/2] – the array index of the parent
      2i+1 – the array index of the left child
      2i+2 – the array index of the right child
   (iii) Heaps – complete binary trees, where parents are no smaller than their children (textbook pages 457 and 518).
   (iv) Making the heap (textbook page 636).
      void make_heap(int a[], int n)
      {
         int k,i;
         for(i=1; i<n; i++)
         {
            k=i;
            while(a[k] is bigger than its parent)
               swap a[k] with its parent and reset k to be the index of the parent;
         }
      }
   (v) Reheapification downward (textbook page 637).
      void reheapify_down(int a[], int n)
      {
         int k=0;
         while(a[k] is smaller than one of its children inside a[0],…,a[n-1])
            swap a[k] with its largest child and reset k to be the index of this child;
      }
   (vi) Heap Sort algorithm (textbook page 635).
      void heapsort(int a[0], int n)
      {
         int k;
         make_heap(a,n);
         for(k=n-1;k>0;k--)
         {
            swap a[0] and a[k];
            reheapify_down(a,k);
         }
(vii) Heap sort – an example

The original data before sorting:
8 7 6 5 2 3 4 11 9 10 12 1

make_heap(a,12)

\[
\begin{array}{c}
\text{swap 3 times} \\
\end{array}\
\]

\[
\begin{array}{c}
\text{swap 2 times} \\
\end{array}\
\]

\[
\begin{array}{c}
\text{swap 2 times} \\
\end{array}\
\]
array representation:
12 11 6 8 10 3 4 5 7 2 9 1

swap a[0] and a[11]
1 11 6 8 10 3 4 5 7 2 9 12

reheapify_down(a,11)

array representation:
11 10 6 8 9 3 4 5 7 2 1 12
swap a[0] and a[10]

1 10 6 8 9 3 4 5 7 2 11 12

reheapify_down(a,10)

array representation:

10 9 6 8 2 3 4 5 7 1 10 11 12

swap a[0] and a[9]

1 9 6 8 2 3 4 5 7 10 11 12

reheapify_down(a,9)

array representation:

9 8 6 7 2 3 4 5 1 10 11 12

swap a[0] and a[8]

1 8 6 7 2 3 4 5 9 10 11 12

reheapify_down(a,8)
array representation:
8 7 6 5 2 3 4 1 9 10 11 12

swap a[0] and a[7]
1 7 6 5 2 3 4 8 9 10 11 12

reheapify_down(a,7)

array representation:
7 5 6 1 2 3 4 8 9 10 11 12

swap a[0] and a[6]
4 5 6 1 2 3 7 8 9 10 11 12

reheapify_down(a,6)
array representation:

6 5 4 1 2 3 7 8 9 10 11 12

swap a[0] and a[5]

3 5 4 1 2 6 7 8 9 10 11 12

reheapify_down(a,5)

array representation:

5 3 4 1 2 6 7 8 9 10 11 12

swap a[0] and a[4]

2 3 4 1 5 6 7 8 9 10 11 12

reheapify_down(a,4)

array representation:

4 3 2 1 5 6 7 8 9 10 11 12

swap a[0] and a[3]

1 3 2 4 5 6 7 8 9 10 11 12

reheapify_down(a,3)
2. Fast Fourier Transform (this topic will not be included in the final exam)
   The process of computing Fourier transform of an array of size $n=2^k$ requires \((n\log_2 n)/2\) operations of multiplication.

3. Graphs and their traversal algorithms
   (i) Depth First Search
       (v is the initial vertex)
       let \(stack =(v)\);
       while \(stack\) is not empty do
           begin
               \(x := top(stack)\);
               if \(x\) is adjacent to a new vertex \(y\)
                   then add \(y\) at the top of the stack
                   else remove \(x\) from the stack
           end
(ii) Breath First Search

(v is the initial vertex)
let queue = (v);
while queue is not empty do
    begin
        x := front(queue);
        if x is adjacent to a new vertex y
            then add y at the tail of the queue
            else remove x from the queue
    end