Heap
heapsort
1.build max heap
2.Exchange the root node with the last node
3.Heapify again
4.Repeat the above until the heap is empty

priority queue using heap

Maximum	O(1)	Insert	O(log n)
Extract_Maximum	O(log n)	Increase-Key	O(log n)

Hash Table
Open addressing
- linear probing
- quadratic probing
- double hashing -> close to uniform hashing

Chaining
- better implementation: BST

Hash function example
- 1.Division: h(k)=k%m
- 2.Mid-square: h(k)=bits(i,i+r-1)(k^2) 改成二進位，取任意段
- 3.shift folding
- 4.digit analysis

Dynamic hashing
- ex. using directories:
  - directory depth = number of bits of the index of the hash table
  - only processing the overflow box, other noew boxes just point to the same address
- ex. directoryless dynamic hashing

Disjoint Set
Implementation
- 1.Array: O(1)find_set
  - union by size
- 2.Tree: O(1) union use array: store the address of the node
  - union by height
  - path compression
- 3.Linklist representation: head,tail,data
  - union by size

Linear Sorting
Break even point is about k*10^6

Counting Sort:
- given k=number of possible input element -> O(n+k)
- after C[] is completed

```
for (j=n;j>=1;j--) {  -> stable
	B[C[A[j]]]=A[j];
	C[A[j]]--;
}
```

Radix Sort:
- LSD first
- time complexity of RadixSort(implemented by counting sort) : O((n + r)*log k/log r)
- when r=10, log k/log r = d = number of digits

RB Tree
Rule:
1.root is black
2.leaf(nil) is black
3.children of a red node are black
4.same black height
nil.left = nil.right = root
$n \leq 2^{(h/2)}-1$

Insert:
let z be red, insert like BST.
//possibly violated rule: 2 & 3
if z is root -> z be black
if z.p is red
//assume z.p=z.p.p.left, else:change all left&right
if uncle is red -> z.p.p=red, z.p.children=black, check z.p.p.p(let z=z.p.p)
if uncle is black
if z=z.p.right
z=z.p, left_rotate(z)
right_rotate(z.p.p) //note: wont have to check again

Delete:
<-????
define x,y,z as in the slide
possibly violated rule: 2 & 3 & 4, but only when y is black
[draw]

Graph
<u,v> leaves u and enters v in digraph
simple path : vertice distinct
strongly connected: for every distinct (u,v),
there is a directed path from v to u and another from u to v
representation:
adjacency matrix: $O(V^2)$space
adjacency list: $O(V+E)$space
*inverse adjacency list

DFS
$O(V+E)$ using adjacency list
mark v.pi (its parent) to build a DFtree/forest
Edge
tree edge: in the tree
back edge: to a black vertex
forward edge & cross edge: only in digraph

B-tree
[draw]
keys:non-increasing order
Minimum degree of B-tree: $t>=2$
Every node other than root have at least t-1 keys -> t children
Every node can have at most 2t-1 keys -> 2t children (In this case, this node is full)
height <= log ((n+1)/2)/log t

FFT
pointwise multiplication: find 2n points -> $O(n)$
Evaluation & Interpolation
Assumption: n is a power of 2.