Lecture 1

① Order of magnitude: $O(n) < O(n\log_2 n) < O(n^2) < O(n^3) < O(1.5^n) < O(2^n) < O(n!) < O(n^n)$

② 1) $T(n)$ is $O(f(n))$ if there exist constants $c>0$ and $n_0 \geq 0$ such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$. $O(f(n))$: $T(n) \leq$

2) $T(n) = \Omega(f(n))$ if $T(n) \geq c \cdot f(n)$ $\Omega(f(n))$: $T(n) \geq$

3) $T(n) = \Theta(f(n))$ if $c_1 f(n) \leq T(n) \leq c_2 f(n)$ $\Theta(f(n))$: $\leq T(n) \leq$

4) $T(n)$ is $o(f(n))$ if for any constant $c>0$, there exists $n_0 \geq 0$ such that $T(n) < c \cdot f(n)$ for all $n \geq n_0$ $o(f(n))$: $T(n) <$

5) $T(n)$ is $\omega(f(n))$ if $T(n) > c \cdot f(n)$ $\omega(f(n))$: $T(n) >$

Lecture 2

① polynomial: n^4

② $\lg n = \log_2 n$, $\ln n = \log_e n$, $\lg^k n = (\lg n)^k$, $\lg\lg n = \lg(\lg n)$

Exponential: 4^n

$\log x^y = y \log x$

$\log xy = \log x + \log y$

$\log \frac{x}{y} = \log x - \log y$

Polylogarithmic: $(\log n)^4$

$a^{\log_b x} = x^{\log_b a}$

$\log_b x = \frac{\log_a x}{\log_a b}$

③ $\sum_{k=1}^{n} k = 1+2+\cdots+n = \frac{n \cdot (n+1)}{2}$

$\sum_{k=0}^{n} x^k = 1+x+x^2+\cdots+x^n = \frac{x^{n+1}-1}{x-1}$ $(x \neq 1)$

$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$ $(0<x<1)$

$\sum_{k=1}^{n} \log k = n\log n - n$

$\sum_{k=1}^{n} \frac{1}{k} = 1+\frac{1}{2}+\cdots+\frac{1}{n} \approx \ln n$

$\sum_{k=1}^{n} k^p = 1^p+2^p+\cdots+n^p = \frac{1}{p+1} n^{p+1}$

④ Iteration Method: 1) step1: Write recursion tree to analysis 2) write inequation to get upper bound and lower bound. 3) Get answeres from upper and lower bound.

⑤ Substitution Method: 1) step1: Make Assumption 2) step2: Proof upper bound: Example: $T(n) = T(\frac{n}{2}) + T(\frac{n}{3}) + n \sqrt{O(n)}$

$T(n) > T(\frac{n}{2}) + T(\frac{n}{3}) + n \leq c_1 \frac{n}{2} + c_1 \frac{n}{3} + n \leq c_1 \cdot n \Rightarrow c_1 \geq 6$

3) Step3: Proof lower bound: $T(n) = T(\frac{n}{2}) + T(\frac{n}{3}) + n \geq c_2 \frac{n}{2} + c_2 \frac{n}{3} + n \geq c_2 n$

⑥ Master's Method: $T(n) = aT(\frac{n}{b}) + f(n)$, $a \geq 1$, $b > 1$, $f(n) > 0$

1) if $f(n) = O(n^{\log_b a - \epsilon})$ for $\epsilon > 0$: $T(n) = \Theta(n^{\log_b a})$

2) if $f(n) = \Theta(n^{\log_b a})$, then: $T(n) = \Theta(n^{\log_b a} \log n)$

3) if $f(n) = \Omega(n^{\log_b a + \epsilon})$ for $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some $c<1$ and all sufficiently large n: $T(n) = O(n^{\log_b a + \epsilon}) = \Theta(f(n))$

Lecture 3

① Internal Sort: The data to be stored is all stored in the computer's main memory.

External Sort: Some of the data to be sorted might be stored in some external, slower memory.

In-place Sort: The amount of extra place required to sort the data is constant with the input size

② Stability: A stable sort preserve relative (original) order of records with equal keys. [同key值的序列元素在排序前后保持相对顺序]

③ Loop invariant: 1) Initialization: True prior to first iteration of the loop 2) Inductive: True before an iteration and before the next iteration 3) when loop terminate, loop invariant help us proove that algorithm is correct.

for insertion Sort: at the start of each iteration of the loop, the $A[1,\cdots,j-1]$ consists of element original in $A[1,\cdots,j-1]$, but in sorted order

④ Insertion Sort

```
for j=2 to A.length
    key = A[j]
    i = j-1
    while i>0 and A[i] > key
        A[i+1] = A[i]
        i = i-1
    A[i+1] = key
```

stable & in place

Bubble Sort

```
for i=1 to A.length:
    for j = A.length to i+1
        if A[j] < A[j-1]:
            swap(A[j], A[j-1])
```

Stable & in place

Lecture 4

① Tree: Connected (No separation) acyclic (No cycle), undirected

② Forest: acyclic, undirected graph, possibly disconnected.

③ Rooted Tree: 1) Ancestor of node X: any node on the path from root→X 2) Descendant of node X: any node with X as its ancester 3) Parent of node X: node immediately before X on path from root. 4) Children of node X: any node with X as its parents 5) sibling of X: nodes sharing parent with X 6) Leaf/external node: without children 7) Internal node: with at least one child.

④ 1) Degree of X: number of children. 2) Depth of X: length of the simple path from root to X 3) level of a tree: all nodes at the same depth 4) height of X: length of the longest simple path from X downward to some leaf node 5) height of a tree: height of root

⑤ 1) Binary tree: child number: 0,1,2. No restriction on level 2) Full binary tree: child number: 0,2. No restriction on level. 3) Complete binary tree: A binary tree in which all leaves are on the same level and number of CH: 0,2

⑥ Property: Binary tree: 1) There are at most 2^l nodes at level l of a binary tree 2) A binary tree with depth d has at most $2^{d+1}-1$ nodes 3) A binary tree with n nodes has depth at least $\lfloor \lg n \rfloor$ $n \leq \sum_{i=0}^{d} 2^i = 2^{d+1}-1$

⑦ Heap: 1) Nearly complete binary tree (Except last layer), fill from left to right 2) for any node X: Parent(X) ≥ X 3) left child of node i: $A[2i]$; right $= A[2i+1]$ 4) Parent of node i = $A[\lfloor i/2 \rfloor]$ 5) Heapsize[A] ≤ length[A] 6) The element in the subarray $A[(\lfloor n/2 \rfloor + 1), \cdots, n]$ are leaves.

⑧ Heap Operation: 1) Max-heapify: $O(\lg n)$ 2) Build-Max-Heap: $O(n)$ 3) HeapSort: $O(n \lg n)$

```
Max-Heapify(A, i)
l = left(i), r = right(i)
if (l <= heap_size(A) & A[l] > A[i]):
    largest = l;
else: largest = i;
if (r <= heap_size(A) & A[r] > A[largest])
    largest = r;
if (largest != i):
    Swap(A, i, largest);
    Heapify(A, largest);
```

$T(n) = \sum_{i=0}^{h} n_i \cdot h_i = \sum_{i=0}^{h} 2^i (h-i) = \sum_{i=0}^{h} \frac{h-i}{2^{h-i}} 2^h = 2^h \sum_{i=0}^{h} \frac{i}{2^i} \left(k = h-i\right) \leq n \cdot \sum_{k=0}^{\infty} \frac{k}{2^k}$

```
Build-Max-Heap(A)
n = length[A]
for i ← ⌊n/2⌋ down to 1:
    do Max-Heapify(A, i, n)
```

```
HeapSort
Build-MAX-HEAP(A)                    O(n)
for i ← length[A] down to 2
    do exchange A[1] ↔ A[i]
    MAX-Heapify(A, 1, i-1)
```

(n-1)·O(lg n) + O(n) = O(n lg n)

⑨ Operations on Priority Queues: 1) Decrease-key: $O(\lg n)$: Heapify 2) Extract-MAX: remove and return element with largest key: $O(\lg n)$ Extract max and then put A[n] leave → A[1] 3) Maximum: $O(1)$ 4) Increase-key: increase key: $O(\lg n)$: Exchange with Parent if larger. 5) Insert: $O(\lg n)$: Insert to the leaf and heap-increase-key Use Max-Heap

⑩ Quick Sort: in place; Average $O(n\log n)$; Worst $O(n^2)$ [Hoare's Partition]

Quick-Sort(A,p,r):
if p<r
then q ← Partition(A,p,r)
QuickSort(A,p,q)
QuickSort(A,q+1,r)

$T(n) = T(q)+T(n-q)+f(n)$
$= T(q)+T(n-q)+n$

Partition: (A,p,r) (pivot: first)
x ← A[p]; i ← p-1; j ← r+1
while True:
do repeat: j ← j-1; until A[j] ≤ x
do repeat: i ← i+1; until A[i] ≥ x
if i<j: then exchange A[i] ↔ A[j]
else: return j
Running Time: $O(n)$; $n = r-p+1$

Lecture 5:

Improve Quick Sort: pivot (first element)

① Randomizing Sorting: At each step exchange element A[p] with an element chosen at random for A[p,...,r]

② Lomuto's Partition with Randomizing Sorting: (pivot: last element)

Algorithm: Randomize-Quick Sort(A,p,r)
if p<r
then: q ← Randomized-Partition(A,p,r)
Randomized-QuickSort(A,p,q-1)
Randomized-QuickSort(A,q+1,r)
$O(n\log n)$

Algorithm: Partition(A,p,r)
x ← A[r]; i ← p-1
for j ← p to r-1:
do if A[j] ≤ x:
then i ← i+1;
exchange A[i] ↔ A[j]
exchange A[i+1] ↔ A[r]; return i+1

③ Analysis of Randomized Sorting with Lomuto's Partition:
$E[X] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} Pr\{z_i \text{ is compared to } z_j\} = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{2}{j-i+1} < \sum_{i=1}^{n-1}\sum_{k=1}^{n}\frac{2}{k+1}$
$= \sum_{i=1}^{n-1} O(\lg n) = O(n\lg n)$

④ Theorem: Any decision tree that sort n elements has height $\Omega(n\lg n)$
$n! \le 2^h \Rightarrow \lg(n!) \le h \Rightarrow n! \ge (\frac{n}{e})^n \Rightarrow h \ge \lg(\frac{n}{e})^n = n\lg n - n\lg e = \Omega(n\lg n)$

⑤ Counting Sort: Sort n integers which are in the range [0,...,r], r=O(n)

Alg. Counting-Sort(A,B,n,k)
for i ← 0 to r
do C[i] ← 0
for j ← 1 to n
do C[A[j]] ← C[A[j]]+1
for i ← 1 to r
do C[i] ← C[i]+C[i-1]
for j ← n downto 1:
do B[C[A[j]]] ← A[j]
C[A[j]] ← C[A[j]]-1

⑥ Radix Sort: d=θ(1), k=O(n)
Sorting looks at one column at a time: 1) For d digit number, Sort the least significant digit first, using stable algorithm. 2) Continue sorting on the next significant digit, (stable sort) until all digits have been sorted
690
↑↑↑
3 2 1
$O(d(n+k))$

Sort	Time	Best	Worst	Space	In-place	Stable
Bubble	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	✓	✓
Insertion	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	✓	✓
Merge	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$	×	✓
Quick	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(\log n)$	✓	×
Heap Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(1)$	✓	×
Counting	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(k)$	×	✓
Radix	$O(n\cdot k)$	$O(nk)$	$O(nk)$	$O(n+k)$	×	✓

① Direct Addressing Operation: Search, Insert, Delete: $O(1)$

② Hash Function h transforms a key into an index in a hash table $T[0,1,\dots,m-1]$. $h: U \to \{0,1,\dots,m-1\}$

③ Collisions: Two or more keys hash to the same slot.

④ Solve Collision using Chain: keep the list not ordered.
Chain-Hash-Insert: insert x at the head of list $T[h(key[x])]$: $O(1)$

⑤ Universal Hashing: Select a hash function at random, from a designed class of functions at beginning of execution.

⑥ Design an Universal hashing: Choose a prime number p. $A=\{0,1,\dots,p-1\}$
m: number of keys. $B=\{1,2,\dots,p-1\}$. hash function: $h_{a,b}(k) = ((ak+b) \bmod p) \bmod m$,
the family of all such function is: $H_{p,m} = \{h_{a,b} : a\in A, b\in B\}$

⑦ Binary Search Tree: left ≤ root ≤ right. In-order: left, root, right
Preorder: root, left, right, Postorder: left, right, root

⑧ Binary Search Tree Basic Operation: Search, Minimum, Maximum, Predecessor, Successor, Insert, Delete. Running Time: average: $O(\log n)$; Worst case: $\theta(n)$ if very unbalanced

⑨ Successor(x)=y, such that key[y] is the smallest key > key[x]
Predecessor(x)=y, such that key[y] is the biggest key < key[x].

Alg: Tree-Successor(x)
if right[x] ≠ null:
then return tree-Min(right[x])
y ← p[x]
while y ≠ null and x = right[y]
do x ← y
y ← p[y]
return y

Case 1: left(x) is non-empty:
predecessor(x) = the max in left(x)
Case 2: left(x) is empty:
1) Go up the tree until current node is right child.
2) if can't go further, x is smallest.

⑩ Insertion: Start from root and go down: $O(h)$

1. (**8 points**) **True or False**

   (a) (**2 points**) **T or** (F) $f(n) + g(n) = O(min\{f(n), g(n)\})$;

   (b) (**2 points**) (T) **or F** $\log n! = \Omega(\log n^n)$;

   (c) (**2 points**) **T or** (F) Randomized quick-sort has lower worst-case running time than quick-sort;

   (d) (**2 points**) (T) **or F** For any array with n elements, one can always find the i-th smallest element within $O(n)$ time.

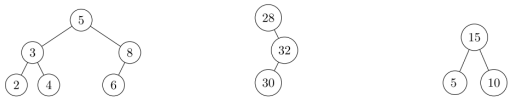
2. (**6 points**) Consider the following sorting algorithms:
   A: InsertionSort; B: MergeSort; C: QuickSort; D: HeapSort.

   (a) (**2 points**) Which of the above sorting algorithms run in worst-case time $O(n \log n)$?
   Circle all that apply: A (B) C (D) None

   (b) (**2 points**) Which of the above sorting algorithms can run even better than $O(n \log n)$ in the best case?
   Circle all that apply: (A) B C D None

   (c) (**2 points**) Which of the above sorting algorithms are stable?
   Circle all that apply: (A) (B) C D None

3. (**6 points**) Consider the following trees:



Tree **A** Tree **B** Tree **C**

   (a) (**2 points**) Which are *binary search trees*?
   Circle all that apply: (A) (B) C None

   (b) (**2 points**) Which are the first three items in a post-order traversal of tree A?
   Circle one: 2,3,4 5,3,8 2,3,5 5,3,2 (2,4,3) 2,4,6 None

   (c) (**2 points**) Which are *max heaps*?
   Circle all that apply: A B (C) None

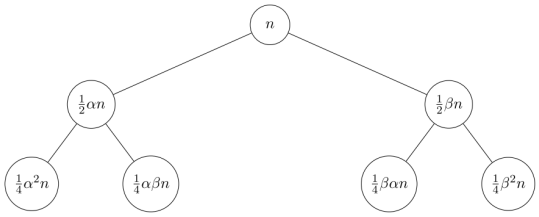
4. (**12 points**) Solve the following recurrences:

   (a) (**4 points**) Use the iteration method to solve $T(n) = \frac{1}{2}(T(\alpha n) + T(\beta n)) + n$, $0.5 < \alpha, \beta < 1$;

   (b) (**4 points**) Use the substitution method to verify your solution for Question 4a.

   (c) (**4 points**) Solve the recurrence $T(n) = 4T(n/2) + n^{2.5}$ using master method.

Answer:



(a)For the kth level, the computation would be $(\frac{1}{2}(\alpha+\beta))^k n$, with summation formula of geometric progression, $\sum_{k=0}^{\inf}(\frac{1}{2}(\alpha+\beta))^k n = \frac{1}{1-\frac{\alpha+\beta}{2}} = \frac{2}{2-\alpha-\beta}n$, since $0.5 < \alpha, \beta < 1$, $T(n) = O(n)$, Recursion Tree (2 pt), correct T(n) (2 pt)
(b)Suppose $T(n) = cn + b$, we substitute T(n) in the original equation, we can get

$$\begin{aligned} T(n) &= \frac{1}{2}(c\alpha n + b + c\beta n + b) + n \\ &= (\frac{1}{2}c\alpha + \frac{1}{2}c\beta + 1)n + b \\ &\leq cn + b \\ &= O(n) \end{aligned}$$

which proves that T(n) = O(n)
(c) As we can see in the formula above, a = 4, b = 2, $log_b a = 2$ and $f(n) = n^{2.5}$, which is the case 3 of Master theorem, then we can say $T(n) = O(n^{2.5})$

5. (**11 points**) For an unsorted array of $[19, 13, 8, 26, 12, 7, 6, 25, 16, 30, 11]$,

   (a) (**5 points**) If heap-sort is applied, plot the initial max-heap built from the array

   Answer: Max-heap array is $[30, 26, 8, 25, 13, 7, 6, 19, 16, 12, 11]$

   (b) (**3 points**) If quick-sort with Hoare's partition is applied, show the result of the first partition call

   Answer: Pivot= 19, $[11, 13, 8, 16, 12, 7, 6, 25, 26, 30, 19]$, iterators: $j = 7$ and $i = 8$

   (c) (**3 points**) If quick-sort with Lomuto's partition is applied, show the result of the first partition call

   Answer: Pivot= 11, $[8, 7, 6, 11, 12, 13, 19, 25, 16, 30, 26]$, iterator: $i = 4$

6. (**13 points**) Design an iterative algorithm to find the k-th smallest element in an array of n distinct elements (k is a given constant, does not grow with n) with worst-case running time of $O(n)$, you are not allowed to use partition and recursive call,

(c) The average of searching for a random key is:

$$1 + \frac{1}{n}\sum_{k=1}^{n}\frac{n-k}{m} = \frac{n-1}{2m} + 1$$

(**20 points**) A k-way merge operation. Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements.

(a) (**5 points**) Here's one strategy: Using the merge procedure, merge the first two arrays, then merge in the third, then merge in the fourth, and so on. What is the time complexity of this algorithm, in terms of k and n?
Answer:

$$2n + 3n + \cdots kn = \frac{(k+2)(k-1)}{2}n = \Theta(k^2 n)$$

(b) (**8 points**) Design and analyze a more efficient algorithm, using divide-and-conquer, that solves this problem in $O(knlogk)$ time.
Answer: Recursive Pairwise Merge

$$T(k) = 2T(\frac{k}{2}) + kn, \text{in terms of just } k, \text{ not } kn$$

Complexity is $O(knlogk)$, not $O(knlogkn)$

(c) (**7 points**) Design and analyze another efficient algorithm, using min-heap, that solves this problem in $O(knlogk)$ time.
Answer: Take the smallest element from each sorted array, form a min-heap with k elements, do extract-min, that will be the minimum of all k arrays, insert the next larger element from the array will the minimum is taken into the min-heap, do extract-min, get the second minimum of all k arrays, repeat kn times to get all sorted.
Total complexity: $O(k) + nkO(\log k) = O(nk \log k)$.

(a) (**8 points**) write down the pseudo-code of your algorithm

Answer: Since $k = O(1)$, possible solutions include but are not limited to:

i. pass through the whole array once, maintain a set of the k smallest elements seen so far

ii. Use selection sort or bubble sort, but stop at the k^{th} iteration of the outer loop: $O(kn) = O(n)$

iii. Create min-heap from all elements and extract the min value k times: $O(n + k \log n) = O(n)$

(b) (**5 points**) what is the maximum number of comparisons your algorithm will make (exact number, not just in the asymptotic sense), under what situation it will happen?

Answer: For (i), the worst case is that each new element has to be compared with all k smallest elements, so $(n-k)k$.
For (ii), there will be exactly $n-1+\cdots+n-k = kn - k(k+1)/2$ comparisons regardless of the initial array (selection sort and bubble sort have this problem in general).

7. (**13 points**) For Radix sort:

(a) (**8 points**) prove that it is correct;

(b) (**5 points**) prove or disprove that it is stable.

Answer:

(a) (**8 points**) if $x = x_1x_2\cdots x_b > y = y_1y_2\cdots y_b$, let j be the first bit where $x_j > y_j$ (automatically $x_i = y_i$, $\forall 1 \le i < j$), then when Radix sort works on bit j, x will be placed after y; since $x_i = y_i$, $\forall 1 \le i < j$, and sorting algorithm for each bit is stable, then the relative position of x and y won't change for the rest of operations (bits $j-1$ to 1), so x and y finishes with the right ordering.

(b) (**5 points**) if $x = x_1x_2\cdots x_b = y = y_1y_2\cdots y_b$, and x is in front of y before sorting, when we sort each bit, x will be always in front of y (stable sort), so x is still in front of y after sorting.

8. (**11 points**) An array of n distinct keys were inserted into a hash table of size m sequentially over n time slots, suppose chaining was used to resolve collisions, let X_k be the random variable of the number of elements examined when searching for the k-th inserted key (the key inserted in the k-th time slot, $1 \le k \le n$),

(a) (**5 points**) what is the probability mass function of X_k, i.e., calculate $P(X_k = i)$, $i \ge 0$?

(b) (**3 points**) what is the expected value of X_k?

(c) (**3 points**) what is the expected number of elements examined when searching for a key randomly selected from the array?

Answer:

(a) There were $n-k$ keys inserted after the k-th key, each of them has a probability of $1/m$ (1pt) to be inserted into the same chain as the k-th key. Let Y_k be the number of keys positioned before the k-th key in its chain, it is a Bernoulli trial of $n-k$ experiments, and success probability of each trial is $1/m$, so Y_k follows a binomial distribution of $(n-k, \frac{1}{m})$, and $X_k = Y_k + 1$, the p.m.f. of Y_k is

$$P(Y_k = i) = \binom{n-k}{i} \frac{1}{m^i} \left(1 - \frac{1}{m}\right)^{n-k-i}, \quad n-k \ge i \ge 0 (2pt)$$

The p.m.f of X_k

$$P(X_k = i) = P(Y_k = i-1) = \binom{n-k}{i-1} \frac{1}{m^{i-1}} \left(1 - \frac{1}{m}\right)^{n-k-i+1}, \quad n-k+1 \ge i \ge 1 \ (2pt)$$

(b) The average of X_k is $\frac{n-k}{m} + 1$