www.cs.njit.edu/~alexg/courses/cs610/index.html
section C login:cs610 pwd:spring17

asymptotics 近似, 漸進的
asymptotically faster in algorithms means grow faster the big-O ordering of running time? (which actually performs slower)

n ^ (1/ln n) = 2
sum (i * x^i) = x / (1-x) ^ 2
sum (i * (1/2)^i) = 1/2 / (1- 1/2) ^ 2 = 2

first exam next week

	stable	in-place	time	
stupid sort			O(n!(n-1))	
odd & even sort	v	v	theta(n^2)	# best case Theta(n)
selection sort	v	v	theta(n^2)	# select the smallest item each round
insertion sort	v	v	O(n ^ 2)	# insert items into sorted sub-list one by one, best case Theta(n)
merge sort	v	x (tree nodes)	O(n log n)	T(n) = 2 * T(n/2) + n - 1 = n log n - (n-1)

1. lim 10 / 25 = const => theta 0
2. lim (n log n) / n^2 = 0 => o 0
3. lim 2^n / 3^n = 0 => o 0
4. lim 2^log n / 2^3 log n = 0 => o 0
5. lim log (n!) / n log n

(1) O(g(n)): there exist c, n0 where f(n) <= c g(n) for all n >= n0
(2) Upper Omega: there exist c, n0 where c g(n) <= f(n) for all n >= n0
(3) Theta:there exist c1, c2, n0 where c1 g(n) <= f(n) <= c2 g(n) for all n >= n0

methods
(1) interation (recursive tree) 窮舉歸納法
(2) master method => T(n) = a T(n/b) + f(n)
given a > 0, b > 1, f(n) > 0 for all n >= n0

a. O(n ^ (log b (底)(a) - e) for some e > 0
b. Theta(n ^ (log b a) * (log n) ^ k) where k >= 0
c. Omega(n ^ (log b a + e)) where e > 0

log b a => b ^ x = a => 10b ^ x = a * 10 ^ x log (a/b) => 10 ^ x = a/b => 10b ^ x = a/b * b ^ x = a * b ^ (x - 1)

(3) substitution method, guess and check

2/14

tree and heap sort (non-stable)

string code: ASCII (1 byte), Unicode (2 bytes), UTF-8 (1 ~ 4 bytes, indicating the length with the ending of first byte)

Huffman's Compression
Entropy
https://en.wikipedia.org/wiki/Huffman_coding

2/21

exam 2 on 2/28 for algorithms

what's the time complexity to check max heap or min heap property: n/2 (only checking internal nodes not leaf nodes)

```
GT-QS(A, n)
GTQS(A, O, n - 1)
GTQS(A, l, r) // A[l ... r]
if l < r {
        m = GT-Partition(A, l, r) A[l ... m-1, m, m+1 .. r]
        GTQS(A, l, m - 1)
        GTQS(A, m + 1, r)
}
```

```
GT-Partition(A, l, r)
spliter = A [r]
i = l; j = r - 1
while(i <= j) {
        while(i <= j && A[i] <= spliter)
                i++
        while(j >= i && A[j] >= spliter)
                j--
        if(i < j)
                swap(a[i], A[j])
```

}
swap(A[i], A[r])
return i

T(n) = T(i) + T(n - 1 - i) + n

worst case: i = n - 1 => theta (n^2), tree level depth n - 1
best case: i = (n-1)/2 => O (n log n), tree level depth log n
avg case: T(n) = 1/n (sum of all T(n) for n = 0 to n - 1) + n

in avg case, use best case split and worst case split by turns:
first split: i = n - 1
second split: i = (n-1)/2
tree level depth: 2 log n
time complexity: O (n log n)

prove of worst case:
sppose T(n) = O(n^2)
T(1) <= c2 x 1
T(2) <= c2 x 2^2
T(n) <= c2 x n^2

T(n+1) = T(0) + T(n) + n = c2 x n^2 + n <= c2 (n+1)^2, for c2 >= 1, n >= 1

random-GT-Partition:
random = rnd(r,l)
swap(A[r], A[random])
GT-Partition(A, l, r)

worst case prob %: 2/n * 2/(n-1) * ... 2/2 = around 1 / n!

best case complexity and worst case, avg case are the same
50% of the time you can pick T(n) = T(n/4) + T(3/4n) + n = O(n log n)

Ta(n, I): running time of alg a on input I of size n
T_insert(n, sorted) = theta(n)
T_insert(n, reverse_sorted) = theta(n^2)
T_insert(n) = O(n^2) (choose the worst case as upper bond)

O(n log n) >= 最好 algorithms 的 worst case >= log (n!) (取整數最大) = ? : time required for n! leaf nodes and pick the best one using binary search

#practice question: sort 5 keys within 7 comparison worst case, 7 comes from log(5!) (取整數最大)
sort 2 keys and 3 keys, then merge the 2 group

https://en.wikipedia.org/wiki/Comparison_sort

exam 3-ID: 044