Sorting Algorithms

- Merge Sort

Merge Sort

Divided, Recursion and Merge

Algo MergeSort(S):

```
Input: an array S, with n size

Output: a sorted S

if S.size() = 2 then

    if S[0] > S[1] then

        exchange S[0] and S[1] in S

    return S

if S.size() > 2 then

    S1, S2 <- Partition(S, n/2)

    S1 <- MergeSort(S1)

    S2 <- MergeSort(S2)

    S <- Merge(S1, S2)

return S
```

Algo Merge(S1, S2):

```
Input: S1, S2 are 2 arrays

output: A sorted merged array S

i <- 0

j <- 0

k <- 0

create a empty array S, with size |S1| + |S

while i <= |S1|-1 and j <= |S2| - 1 do

    if S1[i] <= S2[j] then

        S[k] <- S1[i]

        i++

        k++

    if S1[i] > S2[j] then

        S[k] <- S2[j]

        j++

        k++

while i <= |S1| - 1 do

    S[k] <- S1[i]

        i++

        k++

while j <= |S2| - 1 do

    S[k] <- S2[j]

        j++

        k++

return S
```

MergeSort uses extra space to store sorted array, so it is outplace

For the number in array with same value, for example S[i] = S[j], and i < j, after MergeSort, S[i'] = S[j'], i' is still less than j'. So it is stable.

Runtime(n log n)

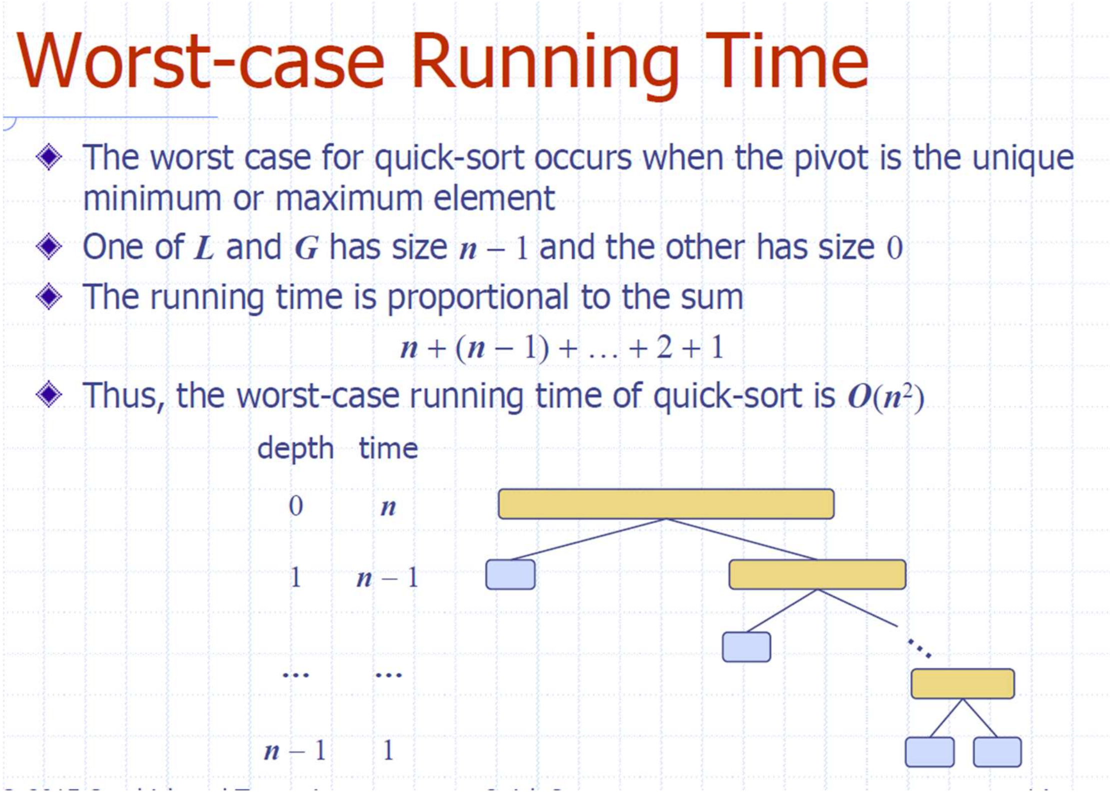
log n is the height of mergeSort tree.

- Quick Sort

Algo QuickSort(S):

```
    if S.size() <= 1 then

        return S

    random choose an element from S as a pivot p

    create 3 empty array L, E, G

    i <- 0

    remove all elements from S, and put them into 3 parts:

    1: L, storing the element that is less than p

    2: E, storing the element that is equal to p

    3: G, storing the element that is greater than p

    L <- QuickSort(L)

    G <- QuickSort(G)

    Merge L, E, G into S

    return S
```

runtime: O(n^2), worst case



worst case runtime O(n^2) -- When we continue to choose the pivot to be the minimum or maximum element on the subarray.

Outplace：L,E,G

不稳定：有多个等于 pivot 的值，可能都排到 pivot 后面去了

- Heap Sort

1. Build Max Heap
2. Swap and Reheap:
    1. Swap the root (maximum element) with the last element in the array, and decrease the heap size by one.
    2. maintain max heap structure and heap order properties (Reheap)
3. Repeat step 2 until done

- Bucket Sort

Bucket Sort

Algo BucketSort(S):

```
Input: S is a sequence of n elements, with integer key range from [0, N-1]

Output: Sorted S

create an empty array of N lists, B

for each element x from S do

    remove x from S

    let k be the key of x

    insert x into B[k]

for i <- 0 to N - 1 do

    while B[i] is not empty do

        remove the element x from B[i]

        insert x at the end of S

return S

runtime: O(n+N)

稳定

out-place
```

- Radix Sort

Radix Sort

Algo RadixSort(S, d):

```
Input: a sequence of n elements S, with d digits

Output: A sorted S

for i <- 1 to d do

    use BucketSort to Sort each elements, according their i-th digit number.
```

runtime: O(d*(n + N))

outplace

稳定

Algorithm quickSelect(S, k):

Input: Sequence S of n comparable elements, and an integer $k \in [1, n]$
Output: The kth smallest element of S

if $n = 1$ **then**
  return the (first) element of S
pick a random element x of S
remove all the elements from S and put them into three sequences:

- L, storing the elements in S less than x
- E, storing the elements in S equal to x
- G, storing the elements in S greater than x.

if $k \leq |L|$ **then**
  quickSelect(L, k)
else if $k \leq |L| + |E|$ **then**
  return x // each element in E is equal to x
else
  quickSelect$(G, k - |L| - |E|)$

Algorithm 9.3: Randomized quick-select algorithm.

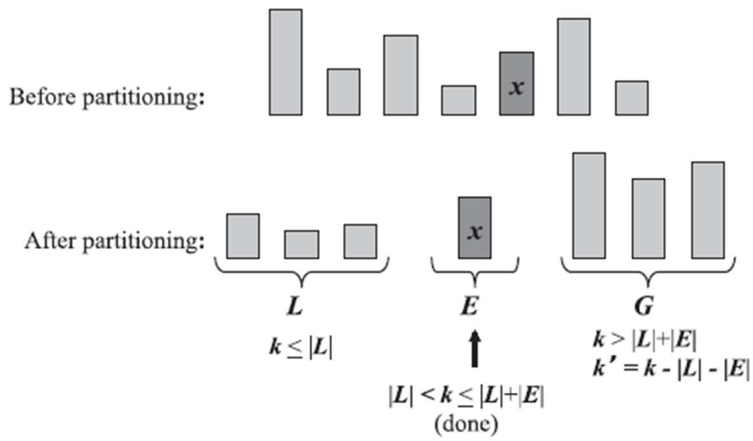


Figure 9.4: A schematic illustration of the quick-select algorithm.

不用 MOM 的 Runtime

Proof by induction: $n=1 \rightarrow T(1)=c$

Inductive hypothesis: Assume $T(n-1) = c(n-1) + T(\frac{3}{4}(n-1)) \leq 4c(n-1)$ is true.

Prove: for n, that $T(n) \leq \cancel{4T(3/4\,n)}\ 4cn$

Conjecture: $T(n) \leq 4cn$

by substitution of claim

$$T(n) = cn + T(3/4\,n) \leq cn + 4c(3/4\,n)$$
$$= cn + 3cn$$
$$T(n) = 4cn \quad \square$$

用 MOM 情况下的 quick Selection Runtime

$$T(n) = c \cdot n + T(3/4\,n) + T(n/5)$$

Runtime for selection algorithm:

$$T(n) = \overset{\text{random tasks}}{c} \cdot n + \underbrace{T(3/4\,n)}_{\substack{\text{going into subarray} \\ \text{(w/ 75\% strategy)}}} + \underbrace{T(n/5)}_{\substack{\text{m.o.m} \\ \text{(selecting pivot)}}}$$

cn: partitioning & housekeeping

Conjecture: $T(n) \leq 20 \cdot cn$

Proof by induction: (see before)

prove: $T(n) = c \cdot n + T(n/5) + T(3/4\,n)$

$$T(n) \leq c \cdot n + \overset{4}{\cancel{20}}c\left(\frac{n}{\cancel{5}}\right) + \overset{5}{\cancel{20}}c\left(\frac{3}{\cancel{4}}\right)n$$

$$T(n) \leq c \cdot n + 4cn + 15cn$$
$$T(n) \leq 20cn$$

Assume n is odd → makes finding median easier. otherwise, requires using ceil or floor.