QuickSort and Others

Data Structures and Algorithms
Andrei Bulatov

Heap Property

A heap is a nearly complete binary tree, satisfying an extra condition

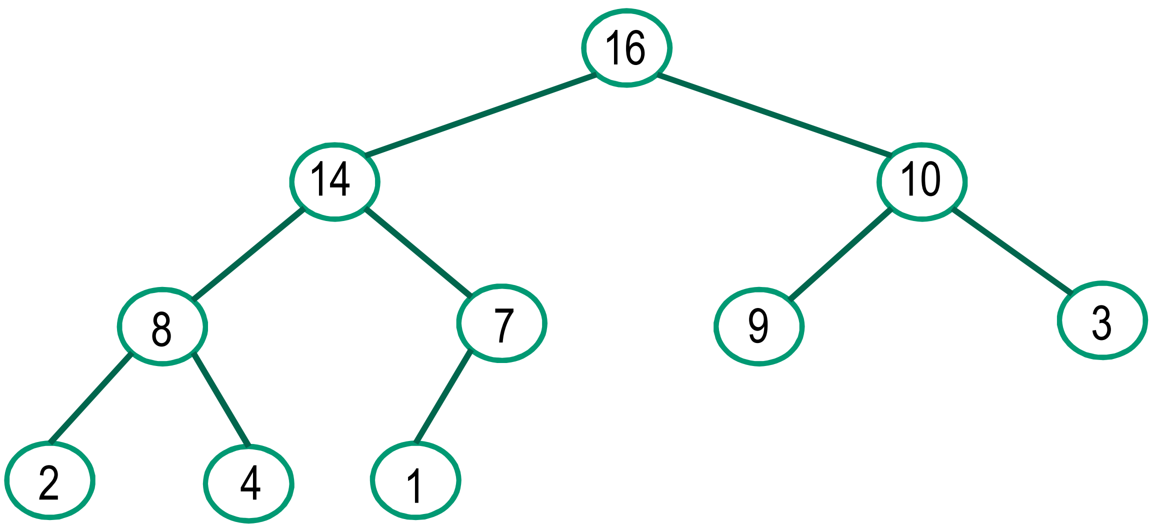
Let Parent(i) denote the parent of the vertex i

Max-Heap Property:

$$Key(Parent(i)) \geq Key(i) \quad \text{for all } i$$

Min-Heap Property:

$$Key(Parent(i)) \leq Key(i) \quad \text{for all } i$$



Insertion

Insert(H,key)

```
set n:=length(n),
set H[n+1]:=key
HeapifyUp(H,n+1)
```

H is almost heap with H[i] too big if decreasing H[i] by a certain amount turns H into a heap

---

```
HeapifyUp(H,i)
if i>1 then
   set j:=parent(i)=⌊i/2⌋
   if Key[H[i]]>Key[H[j]] then
      swap array entries H[i] and H[j]
      HeapifyUp(H,j)
   endif
endif
```

Deletion

Delete(H,i)

```
set n:=length(n),
set H[i]:=H[n]
if Key[H[i]]>Key[H[parent(i)]] then
   HeapifyUp(H,i)
endif
if Key[H[i]]<Key[H[leftChild(i)]] or
  Key[H[i]]<Key[H[rightChild(i)]] then
   HeapifyDown(H,i)
endif
```

H is almost heap with H[i] too small if increasing H[i] by a certain amount turns H into a heap

Deletion (cntd)

```
HeapifyDown(H,i)
set n:=length(H)
if 2i>n then Terminate with H unchanged
else if 2i<n then do
   set left:=2i, right:=2i+1
   let j be the index that minimizes Key[H[left]]
  and Key[H[right]]
else if 2i=n then   set j:=2i
endif
if Key[H[j]]>Key[H[i]] then
    swap array entries H[i] and H[j]
    HeapifyDown(H,j)
endif
```

HeapifyDown: Soundness

Theorem

The procedure HeapifyDown(H,i) fixes the heap property in O(log i) time, assuming that the array H is almost a heap with the key of H[i] too small.

The running time of Deletion is O(log n)

Proof DIY

Building a Heap

```
Build-a-Heap(A)
set n:=length(A)
for i=1 to n do
   set H[i]:=A[i]
   HeapifyUp(H,i)
endfor
```

HeapSort

```
HeapSort(A)
```

Input: `array A`

Output: `sorted array A`

Method:

```
set H:=Build-a-Heap(A)
set n:=length(H)
for i=n downto 1 do
   set A[i]:=H[1]
   set length(H):=length(H)-1
   Delete(H,1)
endfor
```

Priority Queues

A priority queue is a data structure for maintaining a set S of elements, each with associated value called a key.

Priority Queue operations

Insert(S,x) Insert element x into the set S

Maximum(S) Returns the element of S with the largest key

Extract-Max(S) Removes and returns the element of S with the largest key

Increase-Key(S,x,k) Increases the value of element x's key to the new value k, which is at least as large as x's current key value

QuickSort

The idea:

Suppose we have an array A[p..r] to sort

Divide it into two subarrays A[p…q – 1] and A[q + 1…r] such that all elements in the first subarray are smaller or equal to A[q], and all elements in the second subarray are greater or equal to A[q]

Conquer: Recursively sort the subarrays

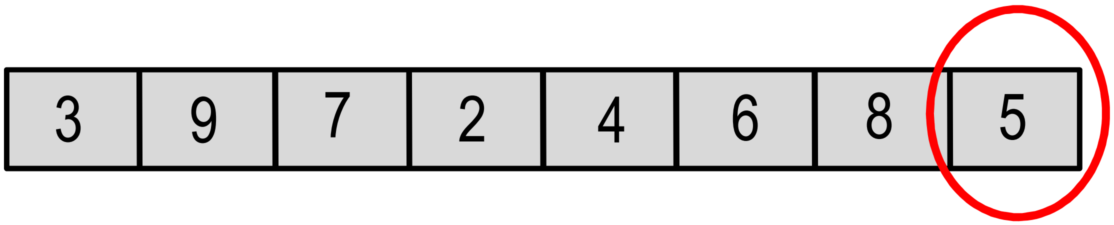
Combine: Don’t have to

QuickSort (cntd)

```
QuickSort(A,p,r)

if p<r then do
   set q:=Partition(A,p,r)
   QuickSort(A,p,q-1)
   Quicksort(A,q+1,r)
endif
```

Partition



pivot element

Rule:

Leave an element as is if it is smaller than the pivot, move it to the right otherwise

Partition: Pseudocode

```
Partition(A,p,r)

set x:=A[r]

set i:=p-1
for j=p to r-1 do
   if A[j]≤x then do
      set i:=i+1
      exchange A[i] and A[j]
   endif
endfor
exchange A[i+1] and A[r]
output i+1
```

Partition: Soundness

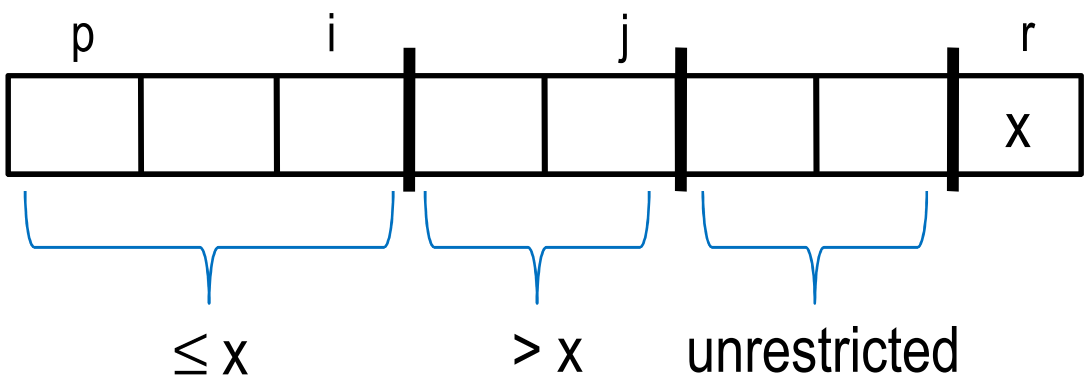
Loop Invariant:

At the beginning of each iteration of the for loop, for any k:

(1) If $p \le k \le i$, then $A[k] \le x$.

(2) If $i + 1 \le k \le j - 1$ then $A[k] > x$.

(3) If $k = r$, then $A[k] = x$



Partition: Soundness (cntd)

Termination:

At termination, $j = r$, therefore every entry of the array is in one of the sets described in the invariant:

less than or equal to x

greater than x

x

The unrestricted region disappears, the array is split

Initialization:

Before the first iteration $i = p – 1$ and $j = p$. Therefore there are no entries between p and i, and also between $i + 1$ and j.

The third condition is obvious

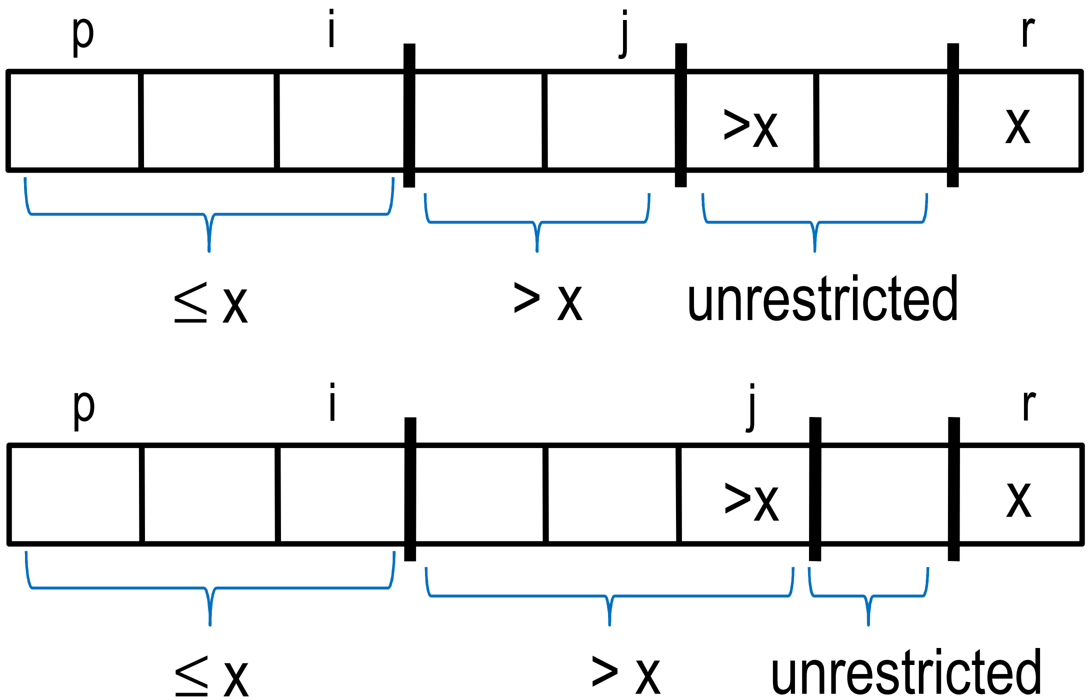
Partition: Soundness (cntd)

Maintenance:

There are two cases depending on the comparison

Case 1. A[j] > x

We only increment j

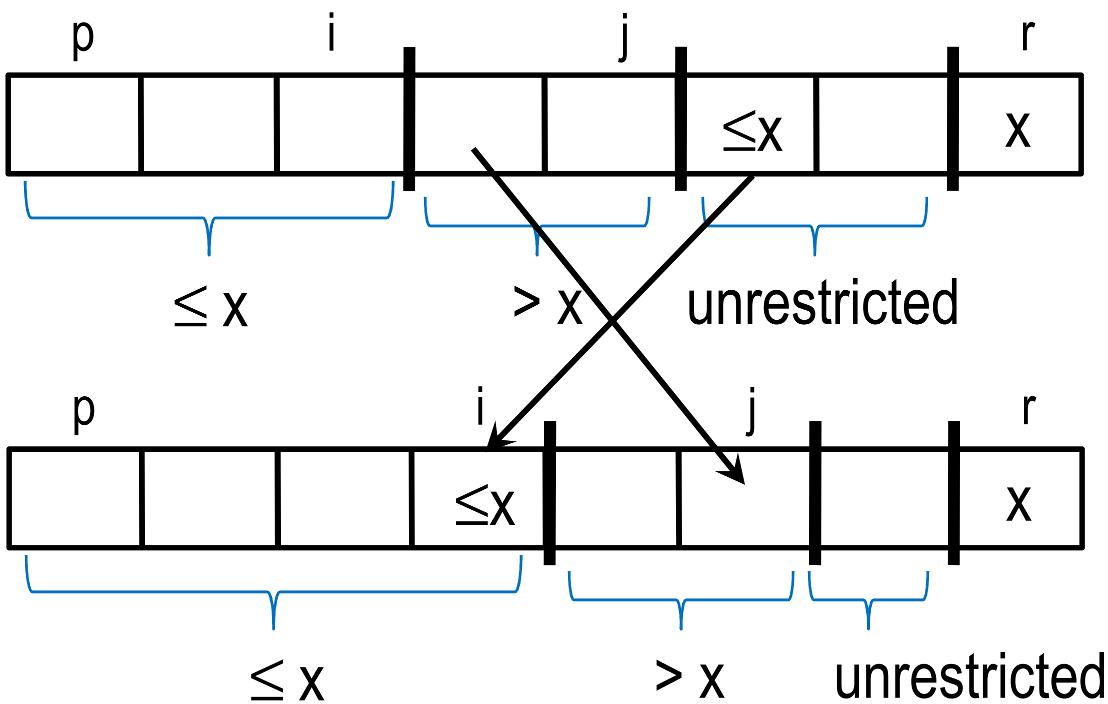


Partition: Soundness (cntd)

Case 2. $A[j] \le x$

i is incremented

A[i] and A[j] are swapped



Partition: Running Time

Best Case:

If every time the array is split into two equal parts

$T(n) = 2\,T(n/2) + Cn$ (Cn is the run. time of Partitioning)

Therefore

$T(n) \in O(n \log n)$

Worst Case:

Every time one of the parts is empty.

Then the depth of recursion is n

Time spent on recursion i is n – i

Therefore $T(n) \in O(n^2)$

Partition: Towards Average Case

Observe that split into two equal parts is not necessary

Suppose the split is into parts containing $n/10$ and $9n/10$ entries

Then

$$T(n) = T(9n/10) + T(n/10) + Cn$$

Recursion tree

As before, $T(n) \in O(n \log n)$

Partition: Towards Average Case (cntd)

Suppose the splits alternate : one into equal parts, another one into an empty part and a part with n – 1 elements

Then

$$T(n) = T(n-1) + Cn$$
$$= 2\,T((n-1)/2) + 2Cn$$

As before, $T(n) \in O(n \log n)$

To examine the average case we need:

a model for inputs distribution

find out what exactly we are going to compute

Probability Reminder

Sample space

Event

Probability

Discrete random variable:

A variable that takes values with certain probability

Example:

The amount of money you win buying a lottery ticket:

there are 1000 tickets, 1 wins \$10000, 10 win \$100, the rest win nothing

$Pr[X = 10000] = 1/1000$, $Pr[X = 100] = 1/100$, $Pr[X = 0] = 989/1000$

Random Variables

Expectation

Let X be a discrete random variable with values $v_1, \ldots, v_k$

Then $E[X] = v_1 \cdot \Pr[X = v_1] + \ldots + v_k \cdot \Pr[X = v_k]$

Example:

E[your win] = 10000·Pr[X = 10000] + 100·Pr[X = 100] + 0·Pr[X = 0]

= 10000· 1/1000 + 100 · 1/100 + 0·989/1000

= 11

One random variable interesting for us is the running time of some algorithm

Properties of Random Variables

Linearity: Let X , Y be discrete random variables, and α a number

Then

E[X + Y] = E[X] + E[Y]

E[αX] = α E[X]

Example:

We flip n fair coins. How many heads do we get on average?

$$X_i = \begin{cases} 1, & \text{if heads on } i\textit{th} \text{ flip} \\ 0, & \text{otherwise} \end{cases}$$

It is called an indicator variable

$$E[X_i] = 1 \cdot \Pr[X_i = 1] + 0 \cdot \Pr[X_i = 0]$$

Let $X = X_1 + \ldots + X_n$ be the total number of heads

$$E[X] = E[X_1 + \ldots + X_n] = E[X_1] + \ldots + E[X_n] = n \cdot \tfrac{1}{2} = \tfrac{n}{2}$$

Homework

Show that the running time of QuickSort is $\Theta(\quad)$ when the array A contains distinct elements and is sorted in decreasing order

What is the running time of QuickSort when all elements of array A have the same value?