Data Structures and Algorithm Analysis (CSC317)

Randomized algorithms (part 2)

Hiring problem - review

- Cost to interview (low c_i)
- Cost to fire/hire… (expensive c_h)

n Total number candidates

m Total number hired

$$O(c_i n + c_h m)$$

$c_i n$: Constant no matter order of candidates

$c_h m$: **Depends on order of candidates!**

Randomized hiring problem(n)

1. randomly permute the list of candidates
2. Hire-Assistant(n)

Hire-Assistant(n)

```
best = 0     //least qualified candidate
for i = 1 to n
    interview candidate i
    if candidate i better than best
        best = i
        hire candidate i
```

Randomized hiring problem(n)

- Instead of worst case
- We would like to know the **cost on average** of hiring new applicants
- X random variable equal to number times new person hired

$$X = \sum_i X_i$$

Randomized hiring problem(n)

- X random variable equal to number times new person hired

$$X = \sum_i X_i$$

Where X_i are indicator random variables:

$$X_i = I\{i\}$$ 1 if candidate i hired
0 if candidate i not hired

For notation, we can drop the I symbol:

$$X_i =$$ 1 if candidate i hired
0 if candidate i not hired

Randomized hiring problem(n)

- X random variable equal to number times new person hired

$$X = \sum_i X_i$$

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i]$$

Linearity expectations

Randomized hiring problem(n)

- X random variable equal to number times new person hired

$$X = \sum_i X_i$$

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i]$$

$$E[X_i] = \Pr(i) = ?$$ Prob candidate i hired

Randomized hiring problem(n)

- X random variable equal to number times new person hired

$$X = \sum_{i=1}^{n} X_i$$

$$* \; E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i]$$

$$E[X_i] = \Pr(i) = \frac{1}{i}$$ Prob candidate i hired

$$E[X] = \sum_{i=1}^{n} \frac{1}{i} = ?$$ Put back in *

Randomized hiring problem(n)

- X random variable equal to number times new person hired

$$X = \sum_{i=1}^{n} X_i$$

$$E[X] = \sum_{i=1}^{n} \frac{1}{i} = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} =$$ Harmonic series

$$\ln(n) + O(1)$$

Randomized hiring problem(n)

- X random variable equal to number times new person hired

$$X = \sum_{i=1}^{n} X_i$$

$$E[X] = \sum_{i=1}^{n} \frac{1}{i} = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} =$$ Harmonic series

$$\ln(n) + O(1)$$

Various ways to prove: One easy way to see:

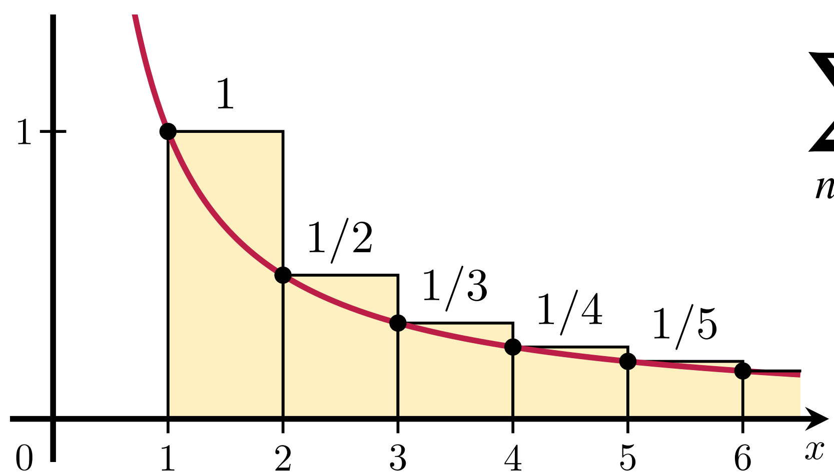
Randomized hiring problem(n)

$$\sum_{i=1}^{n}\frac{1}{i}=\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}=$$

$$\ln(n)+O(1)$$

Harmonic series

Various ways to see:



$$\sum_{n=1}^{k}\frac{1}{n}>\int_{1}^{k+1}\frac{1}{x}dx=\ln(k+1)$$

Randomized hiring problem(n)

- Cost of hiring new candidate C_h
- So on average cost $C_h \ln(n)$
- What about worst case?

Randomized hiring problem(n)

- Cost of hiring new candidate C_h
- So on average cost $C_h \ln(n)$
- Compare to worst case $C_h n$

Quicksort

Input: n numbers
Output: sorted numbers, e.g., in increasing order

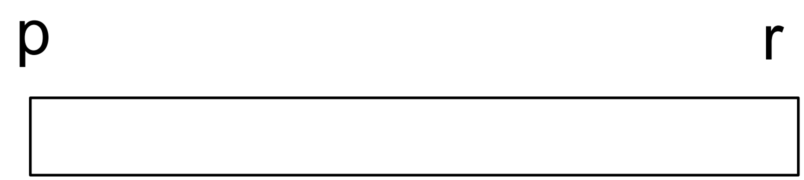
- O(n logn) on average
- Sorts “in place”, unlike Merge sort
- Elegant

Quicksort

Quicksort(A, p, r)

1. If p<r
   2. q = Partition(A,p,r)
   3. Quicksort(A,p,q-1)
   4. Quicksort(A,q+1,r)

Quicksort



Quicksort(A, p, r)

1. **If** p<r
   2. q = Partition(A,p,r)
   3. Quicksort(A,p,q-1)
   4. Quicksort(A,q+1,r)

We can see it is a divide and conquer, but what is special?

Partition

1. Pick pivot element in array A
   (for now, we'll choose last element)
2. Rearrange A so that elements before pivot are smaller, and elements after pivot are larger

Example:

Pivot ↓

A = [8 6 1 3 7 5 2 4]

Partition

Example:

A = [8 6 1 3 7 5 2 4]

After Partition:

A = [1 3 2 4 5 7 6 8]

< pivot   > pivot

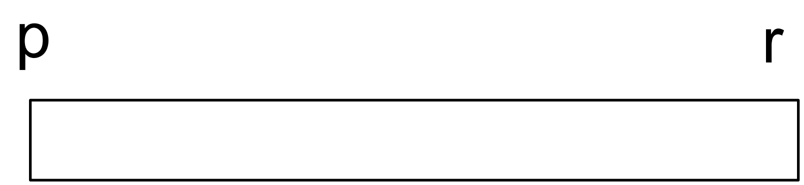
Pivot in its right spot

Partition

Animation:

http://www.cs.miami.edu/~burt/learning/Csc517.101/workbook/partition.html

Partition

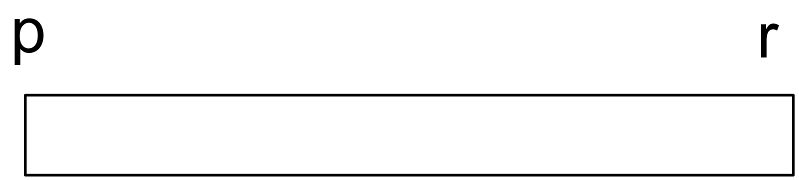


Partition(A,p,r)

```
x=A[ r ]
i=p-1
for j = p to r-1
    if A[ j ] <= x
        i = I + 1;
        swap A[ i ] with A[ j ]
swap A[ i+1 ] with A[ r ]
return i+1
```

O(?)

Partition

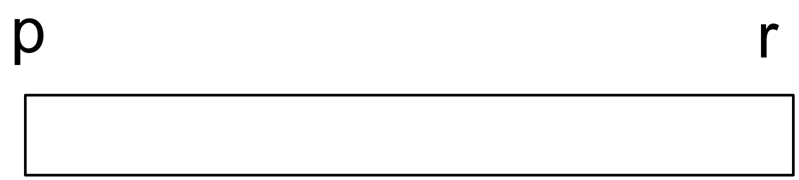


Partition(A,p,r)

1. x=A[r]   pivot
2. i=p-1
3. **for** j = p to r-1
4.   **if** A[j] <= x
5.   i = i + 1
6.   swap A[i] with A[j]
7. swap A[i+1] with A[r]
8. **return** i+1

O(n)
Why?

Partition



Partition(A,p,r)

```
x=A[ r ]    pivot
i=p-1
for j = p to r-1
   if A[ j ] <= x
      i = i + 1
      swap A[ i ] with A[ j ]
swap A[ i+1 ] with A[ r ]
return i+1
```

O(n)
Why? For each j, at most one swap

Partition

Reviewing previous material…

Loop invariant?

Partition

Loop invariant?

i keeps track of pivot location
j keeps track of next element to chk

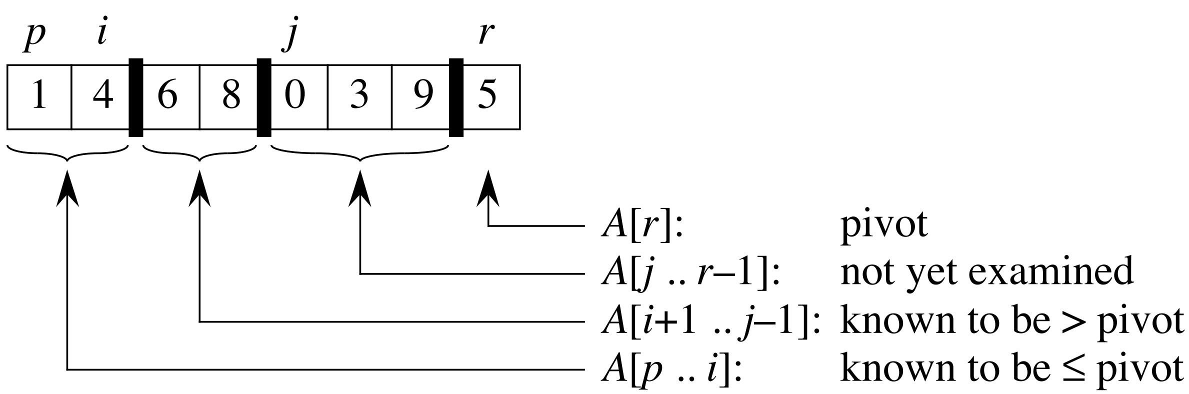
Before the for loop at given j

1. All elements in A[p..i] <= pivot
2. All elements in A[i+1..j-1] > pivot
3. A[r] = pivot

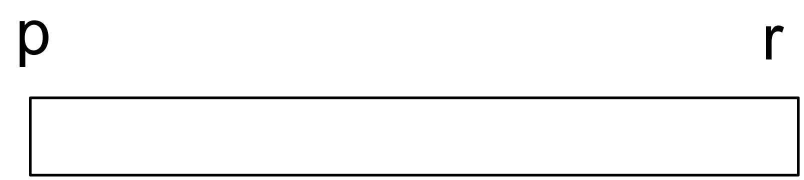
Partition

Loop invariant?

i keeps track of pivot location
j keeps track of next element to chk



Quicksort

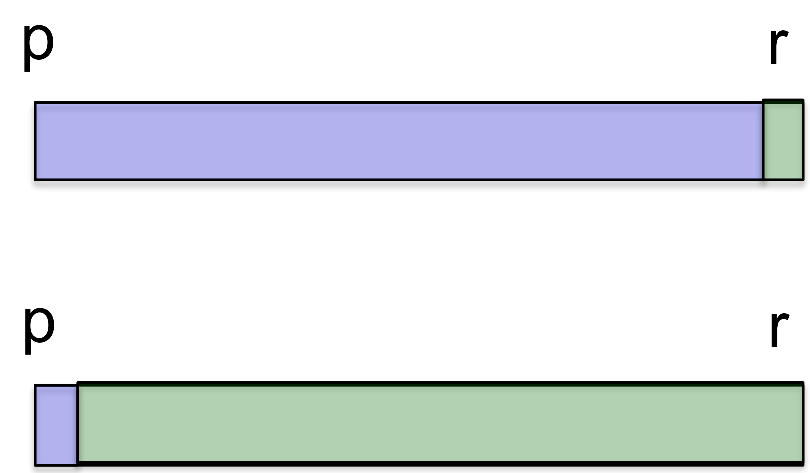


Quicksort(A, p, r)

1. **If** p<r
   2. q = Partition(A,p,r)
   3. Quicksort(A,p,q-1)
   4. Quicksort(A,q+1,r)

When will Partition do a bad job?

Quicksort



Quicksort(A, p, r)

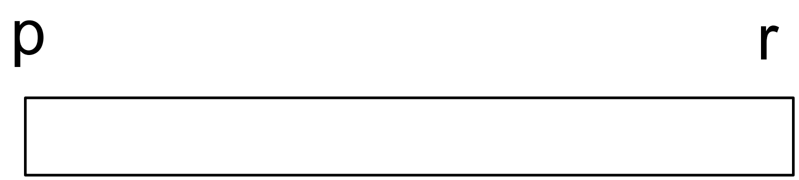
1. **If** p<r
   2. q = Partition(A,p,r)
   3. Quicksort(A,p,q-1)
   4. Quicksort(A,q+1,r)

When will Partition do a bad job?

When the partition is for zero elements versus n-1

$$T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$$

Quicksort

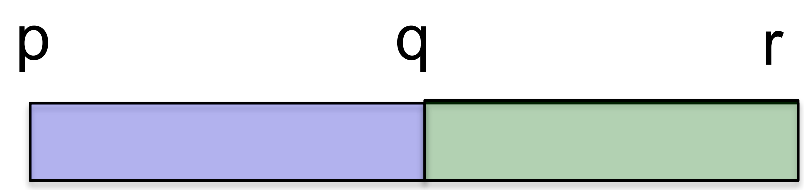


Quicksort(A, p, r)

1. **If** p<r
   2. q = Partition(A,p,r)
   3. Quicksort(A,p,q-1)
   4. Quicksort(A,q+1,r)

When will Partition do a good job?

Quicksort



Quicksort(A, p, r)

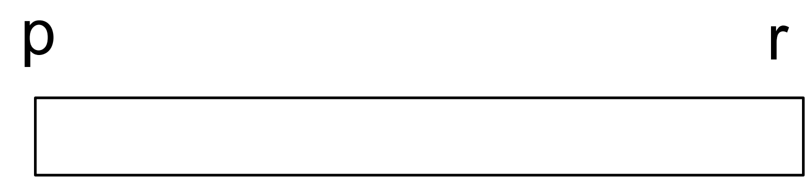
1. **If** p<r
   2. q = Partition(A,p,r)
   3. Quicksort(A,p,q-1)
   4. Quicksort(A,q+1,r)

When will Partition do a good job? When the partition is roughly equal in terms of numbers smaller than and greater than pivot

$$T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$$

(as in Merge sort)

Quicksort

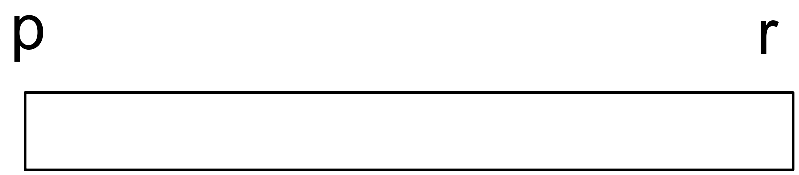


Quicksort(A, p, r)

1. **If** $p<r$
   2. q = Partition(A,p,r)
   3. Quicksort(A,p,q-1)
   4. Quicksort(A,q+1,r)

How do we choose the pivot point??
So that good ON AVERAGE?

Quicksort

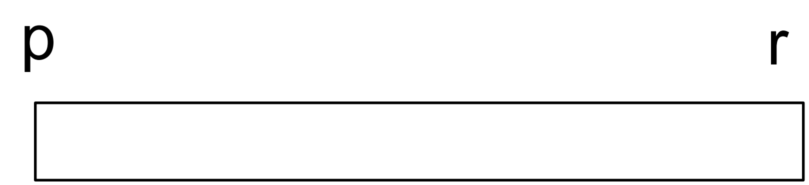


Quicksort(A, p, r)

1. **If** p<r
    2. q = Partition(A,p,r)
    3. Quicksort(A,p,q-1)
    4. Quicksort(A,q+1,r)

KEY APPROACH:
Choose pivot RANDOMLY

Quicksort



Quicksort(A, p, r)

1. **If** p<r
   2. q = Partition(A,p,r)
   3. Quicksort(A,p,q-1)
   4. Quicksort(A,q+1,r)

Why might a random choice be good enough?
We saw that equal split good;
What about a 3 to 1, or 9 to 1 split?

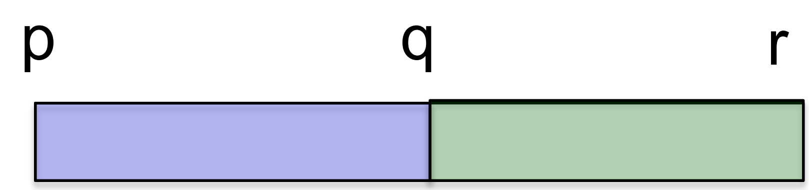
Quicksort

Quicksort(A, p, r)

1. **If** p<r
   2. q = Partition(A,p,r)
   3. Quicksort(A,p,q-1)
   4. Quicksort(A,q+1,r)

What about a 3 to 1, or 9 to 1 split?

Quicksort



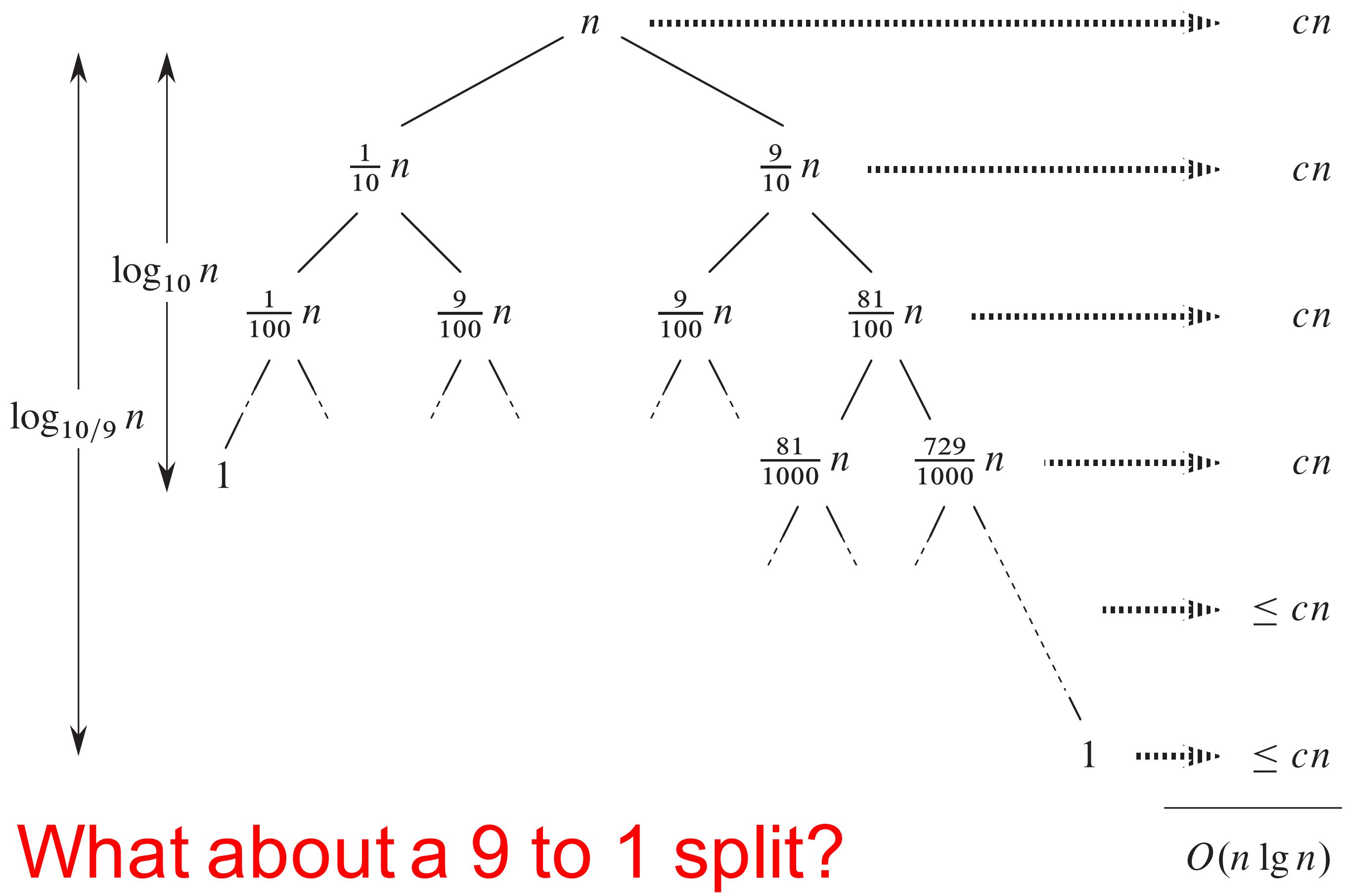
Quicksort(A, p, r)

1. **If** p<r
   2. q = Partition(A,p,r)
   3. Quicksort(A,p,q-1)
   4. Quicksort(A,q+1,r)

What about a 9 to 1 split?

$$T(n) = T(9n/10) + T(n/10) + \Theta(n) = ?$$

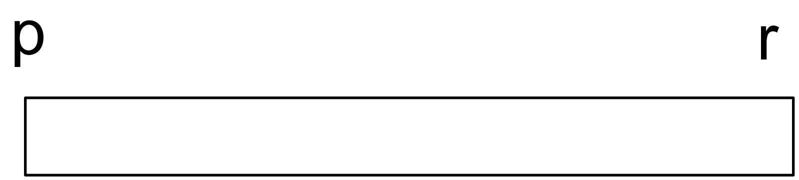
Quicksort



What about a 9 to 1 split?

$$T(n) = T(9n/10) + T(n/10) + \Theta(n) = \Theta(n \log n)$$

Quicksort



Quicksort(A, p, r)

1. **If** p<r
   2. q = Partition(A,p,r)
   3. Quicksort(A,p,q-1)
   4. Quicksort(A,q+1,r)

Why might a random choice be good enough?
Even a 9 to 1, or 3 to 1 split, is O(n log n)

Might not be so hard to get on average??

Quicksort on average run time

- We'll prove that average run time with random pivots for any input array is O(n log n)
- Randomness is in choosing pivot
- Average as good as best case!
- Next class…