CSE 5311 Notes 15: Selected Randomized Algorithms

(Last updated 5/2/16 1:43 PM)

COUPON COLLECTING (KNUTH)

n types of coupon. One coupon per cereal box.

How many boxes of cereal must be bought (expected) to get at least one of each coupon type?

Collecting the n coupons is decomposed into n steps:

Step 0 = get first coupon

Step 1 = get second coupon

Step m = get m+1st coupon

Step n - 1 = get last coupon

Number of boxes for step m

Let p_i = probability of needing *exactly* i boxes (difficult)

$p_i = \left(\frac{m}{n}\right)^{i-1} \frac{n-m}{n}$, so the expected number of boxes for coupon $m + 1$ is $\sum_{i=1}^{\infty} ip_i$

Let q_i = probability of needing *at least* i boxes = probability that *previous* i - 1 boxes are failures (much easier to use)

So, $p_i = q_i - q_{i+1}$

$$
\begin{aligned}
\sum_{i=1}^{\infty} ip_i &= \sum_{i=1}^{\infty} i(q_i - q_{i+1}) \\
&= \sum_{i=1}^{\infty} iq_i - \sum_{i=1}^{\infty} iq_{i+1} \\
&= \sum_{i=1}^{\infty} iq_i - \sum_{i=2}^{\infty} (i-1)q_i \\
&= q_1 + \sum_{i=2}^{\infty} iq_i - \sum_{i=2}^{\infty} iq_i + \sum_{i=2}^{\infty} q_i \\
&= \sum_{i=1}^{\infty} q_i
\end{aligned}
$$

$$q_1 = 1$$

$$q_2 = \frac{m}{n}$$

$$q_3 = \left(\frac{m}{n}\right)^2$$

$$q_k = \left(\frac{m}{n}\right)^{k-1}$$

$$\sum_{i=1}^{\infty} q_i = \sum_{i=0}^{\infty} \left(\frac{m}{n}\right)^i = \frac{1}{1-\frac{m}{n}} = \frac{n}{n-m} = \text{Expected number of boxes for coupon } m + 1$$

Summing over all steps gives

$$\sum_{i=0}^{n-1} \frac{n}{n-i} = n\left(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\right) \le n(\ln n + 1)$$

GENERATING RANDOM PERMUTATIONS

PERMUTE-BY-SORTING (p. 125, skim)

Generates randoms in 1 . . . n^3 and then sorts to get permutation in $\Theta(n \log n)$ time.

Can use radix/counting sort (CSE 2320 Notes 8) to perform in $\Theta(n)$ time.

RANDOMIZE-IN-PLACE

Array A must initially contain a permutation. Could simply be identity permutation: A[i] = i.

```
for i=1 to n
    swap A[i] and A[RANDOM(i,n)]
```

Code is equivalent to reaching in a bag and choosing a number to "lock" into each slot.

Uniform - all $n!$ permutations are equally likely to occur.

Problem 5.3-3 PERMUTE-WITH-ALL

```
for i=1 to n
    swap A[i] and A[RANDOM(1,n)]
```

Produces n^n outcomes, but $n!$ does not divide into n^n evenly.

∴ Not uniform - some permutations are produced more often than others.

Assume n=3 and A initially contains identity permutation. RANDOM choices that give each permutation.

```
1 2 3:  1 2 3   1 3 2   2 1 3   3 2 1
1 3 2:  1 2 2   1 3 3   2 1 2   2 3 1   3 1 1
2 1 3:  1 1 3   2 2 3   2 3 2   3 1 2   3 3 1
2 3 1:  1 1 2   1 3 1   2 2 2   2 3 3   3 1 3
3 1 2:  1 1 1   2 2 1   3 2 2   3 3 3
3 2 1:  1 2 1   2 1 1   3 2 3   3 3 2
```

Asides:

List Ranking - Shared Memory and Distributed Versions (http://ranger.uta.edu/~weems/NOTES4351/09notes.pdf)

Ethernet: http://en.wikipedia.org/wiki/Exponential_backoff

Valiant-Brebner permutation routing

TREAPS (CLRS, p. 333)

Hybrid of BST and min-heap ideas

Gives code that is clearer than RB or AVL (but comparable to skip lists)

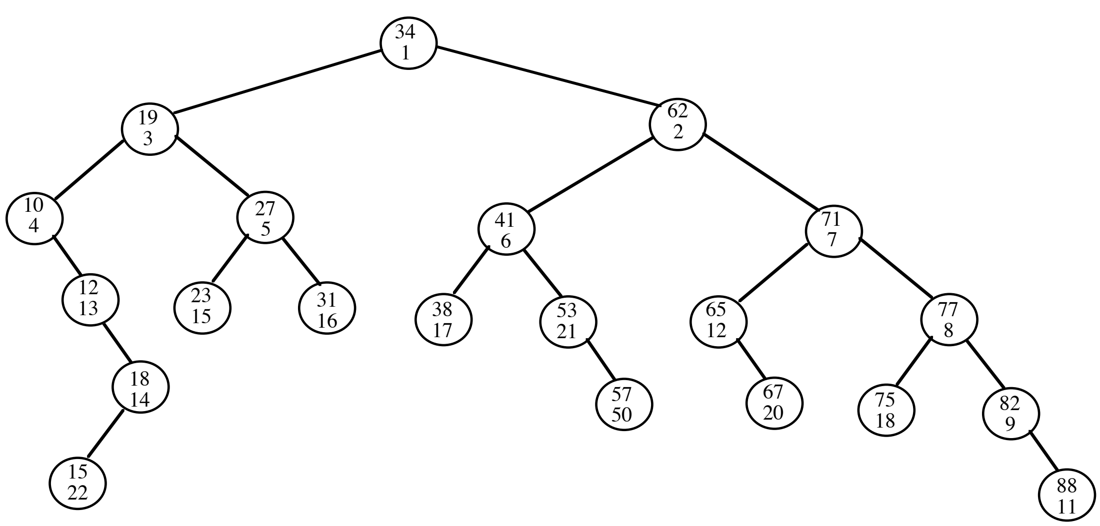
Expected height of tree is logarithmic (2.5 lg n)

Keys are used as in BST

Tree also has min-heap property based on each node having a priority:

Randomized priority - generated when a new key is inserted

Virtual priority - computed (when needed) using a function similar to a hash function



Asides: the first published such hybrid were the *cartesian trees* of J. Vuillemin, "A Unifying Look at Data Structures", *C. ACM 23 (4)*, April 1980, 229-239. A more complete explanation appears in E.M. McCreight, "Priority Search Trees", *SIAM J. Computing 14 (2)*, May 1985, 257-276 and chapter 10 of M. de Berg et.al. These are also used in the elegant implementation in M.A. Babenko and T.A. Starikovskaya, "Computing Longest Common Substrings" in E.A. Hirsch, *Computer Science - Theory and Applications*, LNCS 5010, 2008, 64-75.

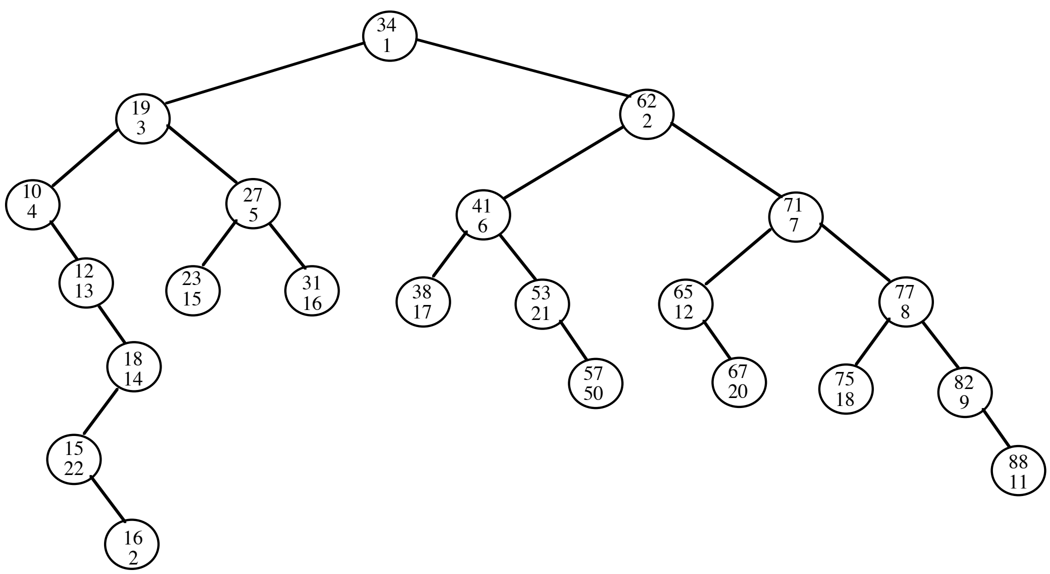
Insertion

Insert as leaf

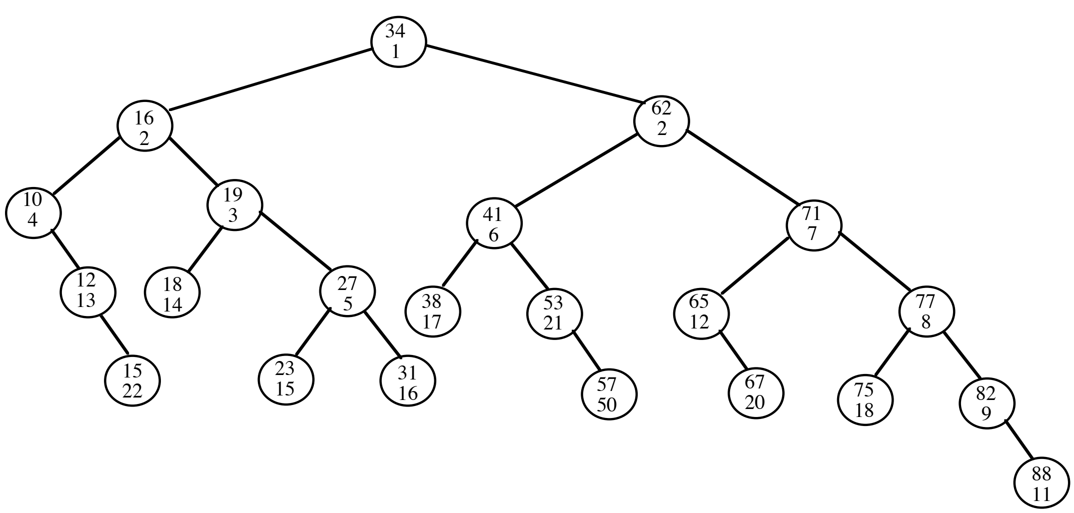
Generate random priority (large range to minimize duplicates)

Single rotations to fix min-heap property

Example: Insert 16 with a priority of 2



After rotations:



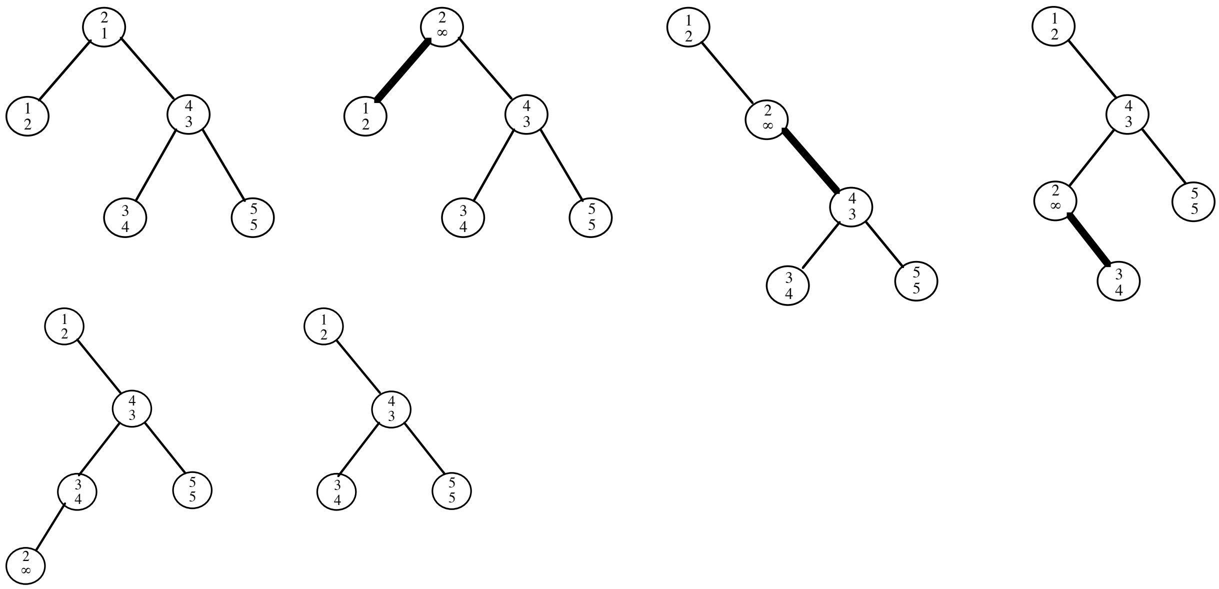
Deletion

Find node and change priority to ∞

Rotate to bring up child with lower priority. Continue until min-heap property holds.

Remove leaf.

Delete key 2:



BLOOM FILTERS (not in book)

`http://dl.acm.org.ezproxy.uta.edu/citation.cfm?doid=362686.362692`

`http://en.wikipedia.org/wiki/Bloom_filter`

M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*, Cambridge Univ. Press, 2005.

m bit array used as filter to avoid accessing slow external data structure for misses

k independent hash functions for the m bit array

Static set of n elements to be represented in filter, but not explicitly stored:

```
for (i=0; i<m; i++)
  bloom[i]=0;
for (i=0; i<n; i++)
  for (j=0; j<k; j++)
    bloom[(*hash[j])(element[i])]=1;
```

Testing if candidate element is possibly in the set of *n*:

```
for (j=0; j<k; j++)
{
  if (!bloom[(*hash[j])(candidate)])
    <Can't be in the set>
}
<Possibly in set>
```

The relationship among *m*, *n*, and *k* determines the *false positive* probability *p*.

Given *m* and *n*, the *optimal* number of functions is $k = \frac{m}{n}\ln 2$ to minimize $p = \left(\frac{1}{2}\right)^k$.

More realistically, *m* may be determined for a desired *n* and *p*: $m = -\frac{n \ln p}{(\ln 2)^2}$ (and $k = \lg \frac{1}{p}$).

What interesting property can be expected for an optimally "configured" Bloom filter?

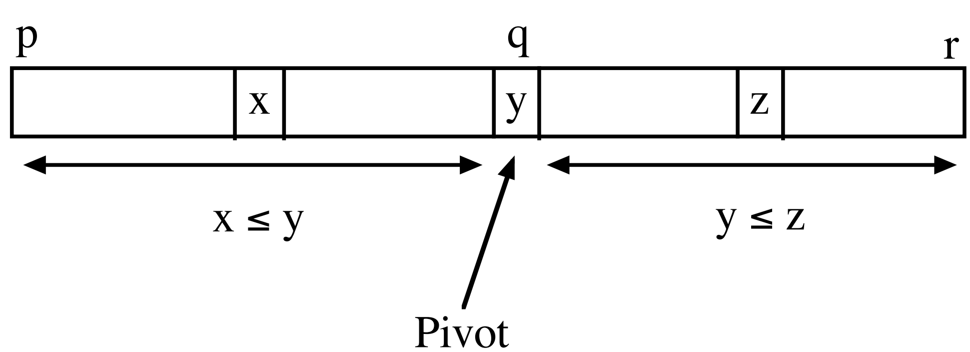
(*m* coupon types, *nk* cereal boxes . . . how many 0's and 1's in bit array?)

8.A. QUICKSORT

Sedgewick 7.1-7.8

Concepts

Idea: Take an unsorted (sub)array and *partition* into two subarrays such that



Customarily, the last subarray element (subscript r) is used as the *pivot* value.

After partitioning, each of the two subarrays, p . . . q – 1 and q + 1 . . . r, are sorted recursively.

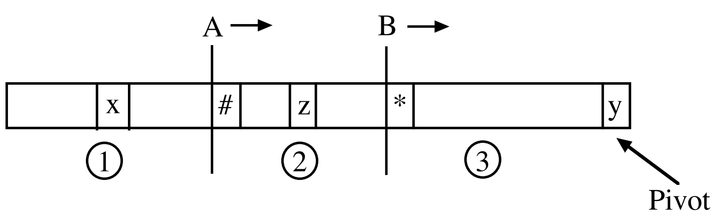
Subscript q is returned from PARTITION (aside: some versions don't place pivot in its final position).

Like MERGESORT, QUICKSORT is a divide-and-conquer technique:

	MERGESORT	QUICKSORT
Divide	Trivial	PARTITION (in-place)
Subproblems	Sort Two Parts	Sort Two Parts
Combine	MERGE (not in-place)	Trivial
Bottom-up possible?	Yes	No

`http://ranger.uta.edu/~weems/NOTES2320/qsortRS.c`

Version 1 (not in Sedgewick): PARTITION (in $\Theta(n)$ time, see `http://ranger.uta.edu/~weems/NOTES2320/partition.c`)



1. Already known to have x ≤ y

2. Already known to have y < z

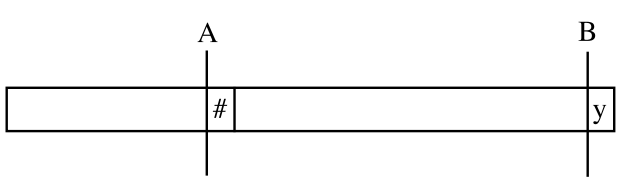
3. Untouched

   y < *: Move B over

   * ≤ y: Swap # & *
   Move B over
   Move A over

A and B can be at the the same position . . .

Termination



Swap # & y to place y in its final position.

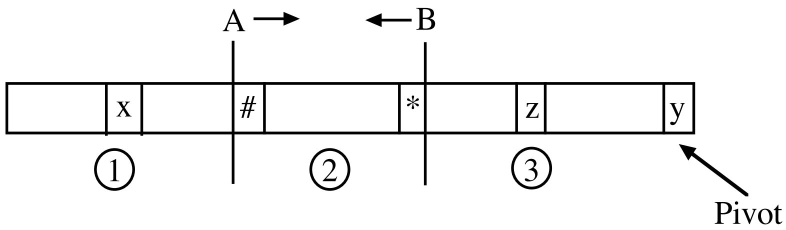
```
int newPartition(int arr[],int p,int r)
// From CLRS, 2nd ed.
{
int x,i,j,temp;

x=arr[r];
i=p-1;
for (j=p;j<r;j++)
  if (arr[j]<=x)
  {
    i++;
    temp=arr[i];
    arr[i]=arr[j];
    arr[j]=temp;
  }
temp=arr[i+1];
arr[i+1]=arr[r];
arr[r]=temp;
return i+1;
}
```

Example:

```
AB 6     3     7     2     8     4     9     0     1     5
A  6   B 3     7     2     8     4     9     0     1     5
   3 A   6   B 7     2     8     4     9     0     1     5
   3 A   6     7   B 2     8     4     9     0     1     5
   3     2 A   7     6   B 8     4     9     0     1     5
   3     2 A   7     6     8   B 4     9     0     1     5
   3     2     4 A   6     8     7   B 9     0     1     5
   3     2     4 A   6     8     7     9   B 0     1     5
   3     2     4     0 A   8     7     9     6   B 1     5
   3     2     4     0     1 A   7     9     6     8   B 5
   3     2     4     0     1   < 5 >   9     6     8     7
```

Version 2 (Sedgewick): Pointers move toward each other (also in $\Theta(n)$ time, see `http://ranger.uta.edu/~weems/NOTES2320/partitionRS.c`)



1 Already known to have x ≤ y

3 Already known to have y ≤ z

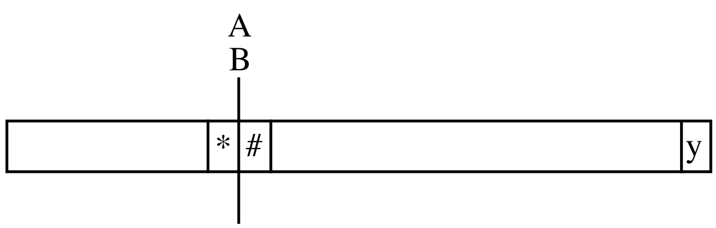
2 Untouched

a # < y: Move A right

b y < *: Move B left

c Swap # and * (unless A and B have collided)

Termination



Swap # & y to place y in its final position.

```
int partition(Item *a,int ell,int r)
{
// From Sedgewick, but more complicated since pointers move
// towards each other.
// Elements before i are <= pivot.
// Elements after j are >= pivot.
int i = ell-1, j = r; Item v = a[r];

printf("Input\n");
dump(arr,ell,r);

for (;;)
  {
    // Since pivot is the right end, this while has a sentinel.
    // Stops at any element >= pivot
    while (less(a[++i], v)) ;
    // Stops at any element <= pivot (but not the pivot) or at the left end
    while (less(v, a[--j])) if (j == ell) break;
    if (i >= j) break; // Don't need to swap
    exch(a[i], a[j]);
  }
exch(a[i], a[r]);     // Place pivot at final position for sort
return i;
}
```

Examples:

```
A  6    3    7    2    8    4    9    0    1    5 B Left positioned
A  6    3    7    2    8    4    9    0    1  B 5  Right positioned
A  1    3    7    2    8    4    9    0    6  B 5  After swap
   1 A  3    7    2    8    4    9    0    6  B 5  Left continues
   1    3 A  7    2    8    4    9    0    6  B 5  Left positioned
   1    3 A  7    2    8    4    9    0  B 6    5  Right positioned
   1    3 A  0    2    8    4    9    7  B 6    5  After swap
   1    3    0 A  2    8    4    9    7  B 6    5  Left continues
   1    3    0    2 A  8    4    9    7  B 6    5  Left positioned
   1    3    0    2 A  8    4    9  B 7    6    5  Right continues
   1    3    0    2 A  8    4  B 9    7    6    5  Right positioned
   1    3    0    2 A  4    8  B 9    7    6    5  After swap
   1    3    0    2    4 A  8  B 9    7    6    5  Left positioned
   1    3    0    2    4 AB 8    9    7    6    5  Pointers collided
   1    3    0    2    4  < 5>   9    7    6    8  Pivot positioned

A  9    8    7    6    5    1    2    3    4    5 B Left positioned
A  9    8    7    6    5    1    2    3    4  B 5  Right positioned
A  4    8    7    6    5    1    2    3    9  B 5  After swap
   4 A  8    7    6    5    1    2    3    9  B 5  Left positioned
   4 A  8    7    6    5    1    2    3  B 9    5  Right positioned
   4 A  3    7    6    5    1    2    8  B 9    5  After swap
   4    3 A  7    6    5    1    2    8  B 9    5  Left positioned
   4    3 A  7    6    5    1    2  B 8    9    5  Right positioned
   4    3 A  2    6    5    1    7  B 8    9    5  After swap
   4    3    2 A  6    5    1    7  B 8    9    5  Left positioned
   4    3    2 A  6    5    1  B 7    8    9    5  Right positioned
   4    3    2 A  1    5    6  B 7    8    9    5  After swap
   4    3    2    1 A  5    6  B 7    8    9    5  Left positioned
   4    3    2    1 A  5  B 6    7    8    9    5  Pointers collided
   4    3    2    1  < 5>   6    7    8    9    5  Pivot positioned
```

QUICKSORT Analysis

Worst Case – Pivot is smallest or largest key in subarray *every time*. (Includes ascending or descending order.) Let $T(n)$ be the number of comparisons.

$$T(n) = T(n-1) + n - 1 = \sum_{i=1}^{n-1} i = \Theta\left(n^2\right)$$

Best Case – Pivot ("median") always ends up in the middle. $T(n) = 2T\left(\frac{n}{2}\right) + n - 1$ (Similar to mergesort.)

Expected Case – Assume all $n!$ permutations are equally likely to occur. Likewise, each element is equally likely to occur as the pivot (each of the n elements will be the pivot in $(n-1)!$ cases). $E(n)$ is the expected number of comparisons. $E(0) = 0$.

$$E(n) = n - 1 + \sum_{i=0}^{n-1} \frac{1}{n}\big(E(i) + E(n-1-i)\big) = n - 1 + \frac{2}{n}\sum_{i=1}^{n-1} E(i)$$

Show $O(n \log n)$. Suppose $E(i) \le ci \ln i$ for $i < n$.

$$E(n) \le n - 1 + \frac{2c}{n}\sum_{i=1}^{n-1} i \ln i \le n - 1 + \frac{2c}{n}\int_1^n x \ln x\, dx \quad \text{[Bound above by integral]}$$

$$= n - 1 + \frac{2c}{n}\left[\frac{1}{2}x^2 \ln x - \frac{x^2}{4}\right]_1^n \quad \text{[From http://integrals.wolfram.com]}$$

$$= n - 1 + \frac{2c}{n}\left(\frac{n^2}{2}\ln n - \frac{n^2}{4} + \frac{1}{4}\right) = n - 1 + cn\ln n - \frac{cn}{2} + \frac{c}{2n}$$

$$\le cn \ln n \text{ for } c \ge 2$$

Other issues: (Sedgewick 7.3-7.5)

Unbalanced partitioning also leads to worst-case *stack depth* in $\Theta(n)$.

Small subfiles - use simpler sort on each subfile or delay until quicksort finishes.

Pivot selection - random, median-of-three

Subfile with all keys equal for version 1 and 2 partitioning?

8.B. Selection and Ranking Using Quicksort Partitioning Ideas

Sedgewick 7.8

Finding kth largest (or smallest) element in unordered table of n elements

(Aside: If k is small, e.g. $O\left(\frac{n}{\log n}\right)$, use a heap.)

Sort everything?

Use PARTITION several times. Always throw away the subarray that cannot include the target.

```
http://ranger.uta.edu/~weems/NOTES2320/selection.c
```

$\Theta\left(n^2\right)$ worst case (e.g. input ordered)

$\Theta(n)$ expected. Let $E(k,n)$ represent the expected number of comparisons to find the kth largest in a set of n numbers. (Assume all $n!$ permutations are equally likely.)

Suppose $n = 7$ and $k = 3$. After 6 comparisons to place a pivot, the 7 possible pivot positions require different numbers of additional comparisons:

1	$E(3,6)$
2	$E(3,5)$
3	$E(3,4)$
4	$E(3,3)$
5	0
6	$E(1,5)$
7	$E(2,6)$

Suppose $n = 8$ and $k = 6$. After 7 comparisons to place a pivot, the 8 possible pivot positions require different numbers of additional comparisons:

1	$E(6,7)$
2	$E(6,6)$
3	0
4	$E(1,3)$
5	$E(2,4)$
6	$E(3,5)$
7	$E(4,6)$
8	$E(5,7)$

Observation: Finding the median is slightly more difficult than all other cases.

$$E(k,n) = n - 1 + \frac{1}{n}\sum_{i=1}^{k-1} E(i, n-k+i) + \frac{1}{n}\sum_{i=k}^{n-1} E(k,i)$$

Show O(n). Using substitution method, suppose $E(i,j) \le cj$ for $j < n$.

$$E(k,n) \le n-1+\frac{1}{n}\sum_{i=1}^{k-1} c(n-k+i)+\frac{1}{n}\sum_{i=k}^{n-1} ci$$

$$= n-1+\frac{c}{n}\sum_{i=1}^{k-1}(n-k+i)+\frac{c}{n}\sum_{i=k}^{n-1} i$$

$$= n-1+\frac{c}{n}\sum_{i=1}^{k-1}(n-k)+\frac{c}{n}\sum_{i=1}^{k-1} i+\frac{c}{n}\sum_{i=k}^{n-1} i$$

$$= n-1+\frac{c}{n}(k-1)(n-k)+\frac{c}{n}\sum_{i=1}^{n-1} i = n-1+\frac{c}{n}(k-1)(n-k)+\frac{c}{n}\frac{(n-1)n}{2}$$

$$\le n-1+\frac{c}{n}\left(\frac{n^2}{4}-\frac{n}{2}+\frac{1}{4}\right)+\frac{c}{2}(n-1) \qquad k=\frac{n+1}{2} \text{ maximizes } \frac{c}{n}(k-1)(n-k)$$

$$= n-1+\frac{cn}{4}-\frac{c}{2}+\frac{c}{4n}+\frac{cn}{2}-\frac{c}{2} = n-1+\frac{3cn}{4}-c+\frac{c}{4n} = cn-\frac{cn}{4}+n-1+\frac{c}{4n}-c$$

$$\le cn \text{ for } c \ge 4$$