CSE 2320 Notes 1: Algorithmic Concepts

(Last updated 8/15/06 3:00 PM)

CLRS, Chapters 1 & 2

Pseudocode Conventions (p. 19-20)

Array Subscripts:

Book: 1 . . . n

Notes/C/Java Code: 0 . . . n – 1

INSERTION SORT:

Concept (`insertionSort.c`)

```
void insertionSort(int *a,int n)
{
int i,j,v;

for (i=1; i<n; i++)
{
  v=a[i];
  j=i;
  while (j>=1 && a[j-1]>v)
  {
    a[j]=a[j-1];
    j--;
  }
  a[j]=v;
}
}
```

Maximum ("worst case") number of times that body of `j`-loop executes for a particular value of `i`?

Maximum number of times that body of `j`-loop executes over entire sort?

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2} = ?$$

Expected ("average") number of times that body of `j`-loop executes for a particular value of `i`?

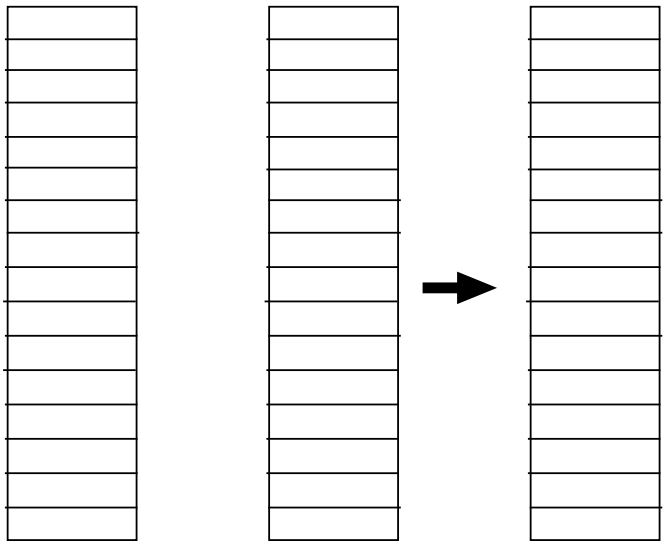
Expected number of times that body of `j`-loop executes over entire sort?

DIVIDE AND CONQUER (Decomposition)

1. Divide into subproblems (unless size allows a trivial solution).
2. Conquer the subproblems.
3. Combine solutions to subproblems.

(Binary) Mergesort (`mergeSort.c`)

1. Split array into two sub-arrays (unless n=1).
2. Call Mergesort recursively for each sub-array.
3. Merge the two ordered sub-arrays. HOW?



```
int merge(int *in1,int *in2,int *out1,int in1Size,int in2Size)
{
int i,j,k;

i=j=k=0;
while (i<in1Size && j<in2Size)
  if (in1[i]<in2[j])
    out1[k++]=in1[i++];
  else
    out1[k++]=in2[j++];
if (i<in1Size)
  for ( ;i<in1Size;i++)
    out1[k++]=in1[i];
else
  for ( ;j<in2Size;j++)
    out1[k++]=in2[j];
return k;
}
```

How are duplicates handled?

[Write body of while-loop with ? : expression. Code for linked lists, files, streams, etc.]

How much work (time) in worse case? (T(n) – a *recurrence*)

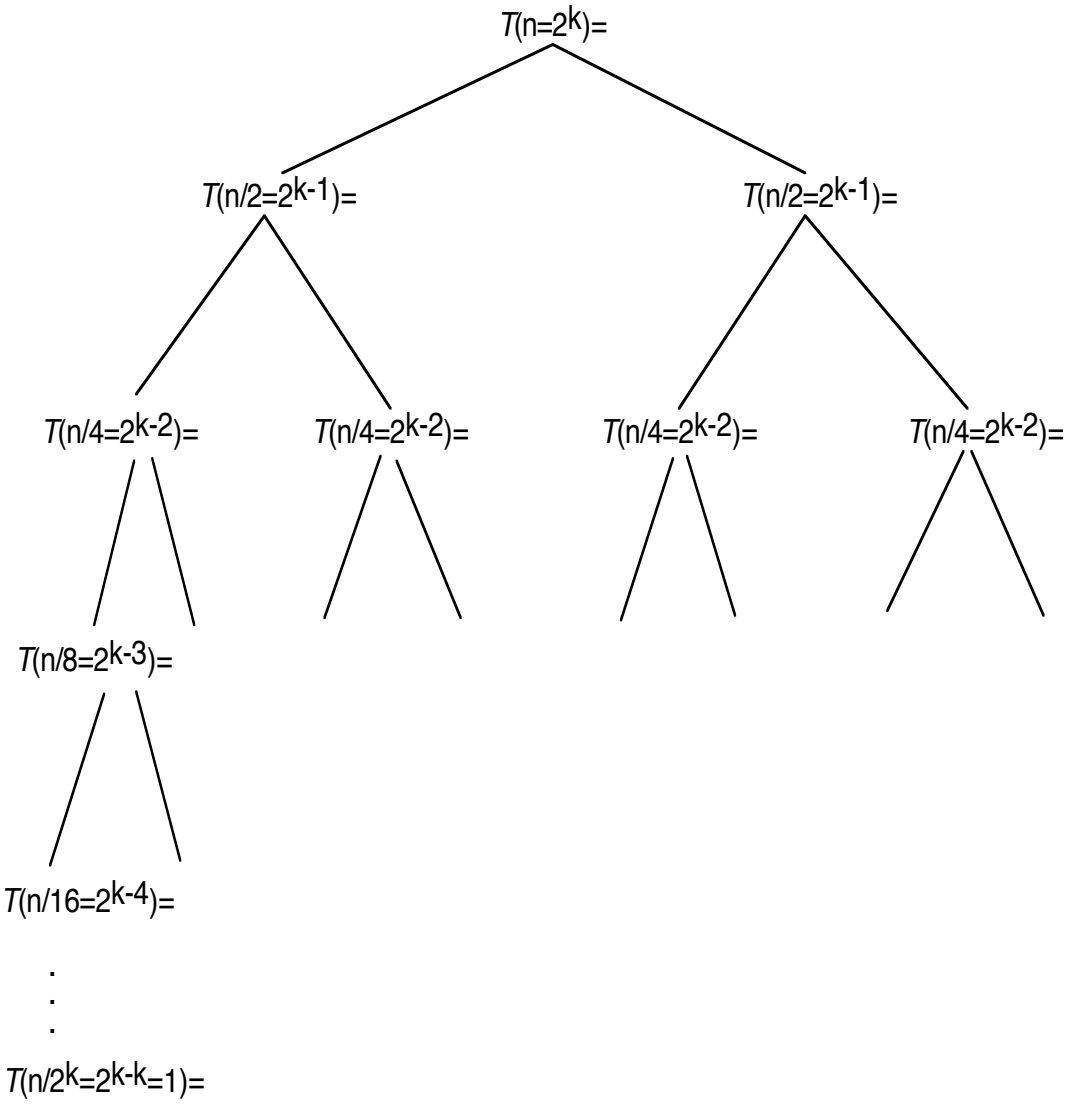
1. Split: n steps. [Can reduce to constant time by computing pointer.]
2. Call recursively:

$$T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right)$$

3. Merge together (n steps)

$$T(n) = c_1 n + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + c_2 n = cn \log? n$$

Recursion Tree



[Don't generalize from this example. More of these later.]

Binary Search

Concept – search *ordered* table in logarithmic time. Consider table with $2^k - 1$ slots.

(binarySearch.c)

```
// Demonstration of everyday binary search.
// No input.  Output is result from searching for each number in -1..24
#include <stdio.h>

int a[10]={0,2,4,6,8,10,12,14,16,18};

int binSearch(int *a,int n,int key)
// Input: int array a[] with n elements in ascending order.
//        int key to find.
// Output: Returns some subscript of a where key is found.
//         Returns -1 if not found.
// Processing: Binary search.
{
  int low,high,mid;
  low=0;
  high=n-1;
// subscripts between low and high are in search range.
// size of range halves in each iteration.
  while (low<=high)
  {
    mid=(low+high)/2;
    if (a[mid]==key)
      return mid; // key found
    if (a[mid]<key)
      low=mid+1;
    else
      high=mid-1;
  }

  return (-1); // key does not appear
}

main()
{
int i;

for (i=(-1);i<25;i++)
  printf("key==%d returned %d\n",i,binSearch(a,10,i));
}
```

```
BSEARCH(3)                Linux Programmer's Manual                BSEARCH(3)

NAME
       bsearch - binary search of a sorted array.

SYNOPSIS
       #include <stdlib.h>

       void *bsearch(const void *key, const void *base, size_t nmemb,
              size_t size, int (*compar)(const void *, const void *));

DESCRIPTION
       The  bsearch() function searches an array of nmemb objects, the initial
       member of which is pointed to by base, for a member  that  matches  the
       object  pointed  to  by  key.   The size of each member of the array is
       specified by size.

       The contents of the array should be in ascending sorted order according
       to the comparison function referenced by compar.  The compar routine is
       expected to have two arguments which point to the key object and to  an
       array  member,  in  that order, and should return an integer less than,
       equal to, or greater than zero if the  key  object  is  found,  respec-
       tively, to be less than, to match, or be greater than the array member.

RETURN VALUE
       The bsearch() function returns a pointer to a matching  member  of  the
       array,  or  NULL  if no match is found.  If there are multiple elements
       that match the key, the element returned is unspecified.

CONFORMING TO
       SVID 3, BSD 4.3, ISO 9899 (C99)

SEE ALSO
       qsort(3)

GNU                               1993-03-29                        BSEARCH(3)
```

(bsearchSearch.c)

```
// Demonstration of bsearch().
// No input.  Output is result from searching for each number in -1..24
#include <stdio.h>
#include <stdlib.h>

int a[10]={0,2,4,6,8,10,12,14,16,18};

int intCompare(const void* xin, const void* yin)
// Used in calls to bsearch()
{
return *((int*) xin) - *((int*) yin);
}

int binSearch(int *a,int n,int key)
// Input: int array a[] with n elements in ascending order.
//        int key to find.
// Output: Returns some subscript of a where key is found.
//         Returns -1 if not found.
// Processing: Binary search using bsearch().
{
  int *mid;

  mid=bsearch(&key,a,n,sizeof(int),intCompare);

  if (mid==NULL)
    return (-1);  // key does not appear
  else
    return mid-a; // ptr arithmetic to get subscript
}
```

```
main()
{
int i;

for (i=(-1);i<25;i++)
  printf("key==%d returned %d\n",i,binSearch(a,10,i));
}
```

(binarySearchRange.c)

```
// Demonstration of special binary searches for tables with duplicates.
// No input.  Output is how many times each number in
// -1 .. 20 appears in a table.
#include <stdio.h>

int a[20]={0,1,1,1,2,4,4,6,6,6,
           10,12,12,12,12,15,15,17,17,18};

int binSearchFirst(int *a,int n,int key)
// Input: int array a[] with n elements in ascending order.
//        int key to find.
// Output: Returns subscript of the first a element >= key.
//         Returns n if key>a[n-1].
// Processing: Binary search.
{
  int low,high,mid;
  low=0;
  high=n-1;
// Subscripts between low and high are in search range.
// Size of range halves in each iteration.
// When low>high, low==high+1 and a[high]<key and a[low]>=key.
  while (low<=high)
  {
    mid=(low+high)/2;
    if (a[mid]<key)
      low=mid+1;
    else
      high=mid-1;
  }
  return low;
}
```

```
int binSearchLast(int *a,int n,int key)
{
// Input: int array a[] with n elements in ascending order.
//        int key to find.
// Output: Returns subscript of the last a element <= key.
//         Returns -1 if key<a[0].
// Processing: Binary search.
  int low,high,mid;
  low=0;
  high=n-1;
// subscripts between low and high are in search range.
// size of range halves in each iteration.
// When low>high, low==high+1 and a[high]<=key and a[low]>key.
  while (low<=high)
  {
    mid=(low+high)/2;
    if (a[mid]<=key)
      low=mid+1;
    else
      high=mid-1;
  }
  return high;
}

main()
{
int i,first,last,key;

printf("-- table --\n");
for (i=0;i<20;i++)
  printf("%3d %3d\n",i,a[i]);
printf("key  first  last  count\n");
for (i=(-1);i<21;i++)
{
  first=binSearchFirst(a,20,i);
  last=binSearchLast(a,20,i);
  printf("%3d   %3d   %3d   %3d\n",i,first,last,last-first+1);
}
}
```

```
-- table --         key  first  last  count
  0    0             -1     0    -1     0
  1    1              0     0     0     1
  2    1              1     1     3     3
  3    1              2     4     4     1
  4    2              3     5     4     0
  5    4              4     5     6     2
  6    4              5     7     6     0
  7    6              6     7     9     3
  8    6              7    10     9     0
  9    6              8    10     9     0
 10   10              9    10     9     0
 11   12             10    10    10     1
 12   12             11    11    10     0
 13   12             12    11    14     4
 14   12             13    15    14     0
 15   15             14    15    14     0
 16   15             15    15    16     2
 17   17             16    17    16     0
 18   17             17    17    18     2
 19   18             18    19    19     1
                     19    20    19     0
                     20    20    19     0
```

`bsearchSearchRange.c` also implements counting of duplicates in logarithmic time, but using a different binary search based on `bsearch()`:

`binSearchFirst()` - Find `i` such that `a[i-1] < key <= a[i]`

`binSearchLast()` - Find `j` such that `a[j] <= key < a[j+1]`