

CSE 3318 Notes 1: Algorithmic Concepts

(Last updated 1/5/24 10:19 AM)

CLRS, Chapters 1 & 2

Pseudocode Conventions (p. 21-24)

Array Subscripts:

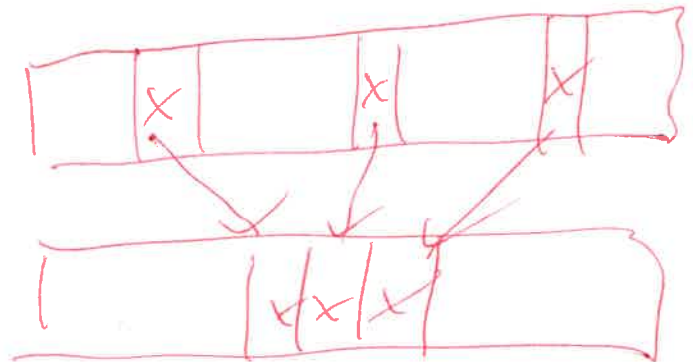
Book: 1...n

Notes/C/Java Code: 0...n-1

1.A. QUADRATIC TIME SORTS:

Selection Sort (CLRS exercise 2.2-2)

```
void selection(Item a[], int ell, int r)
{
    int i, j;
    for (i = ell; i < r; i++)
    {
        int min = i;
        for (j = i+1; j <= r; j++)
            if (less(a[j], a[min]))
                min = j;
        exch(a[i], a[min]);
    }
}
```

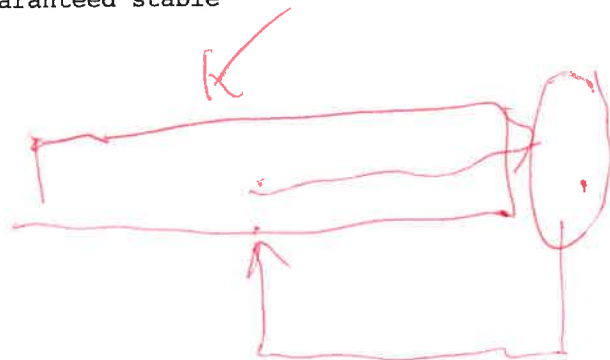


Always uses $\sum_{i=2}^n (i-1) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \approx \frac{n^2}{2}$ comparisons and is not stable (CLRS, p. 210).

(Aside: <https://www.americanscientist.org/article/gausss-day-of-reckoning>)

Insertion Sort (CLRS p.19, <https://ranger.uta.edu/~weems/NOTES3318/insertionSort.c>)

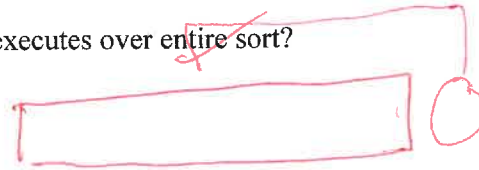
```
void insertionSort(Item *a, int N) // Guaranteed stable
{
    int i, j;
    Item v;
    for (i=1; i<N; i++)
    {
        v=a[i];
        j=i;
        while (j>1 && less(v,a[j-1]))
        {
            a[j]=a[j-1];
            j--;
        }
        a[j]=v;
    }
}
```



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?

1.B. DIVIDE AND CONQUER (Decomposition)

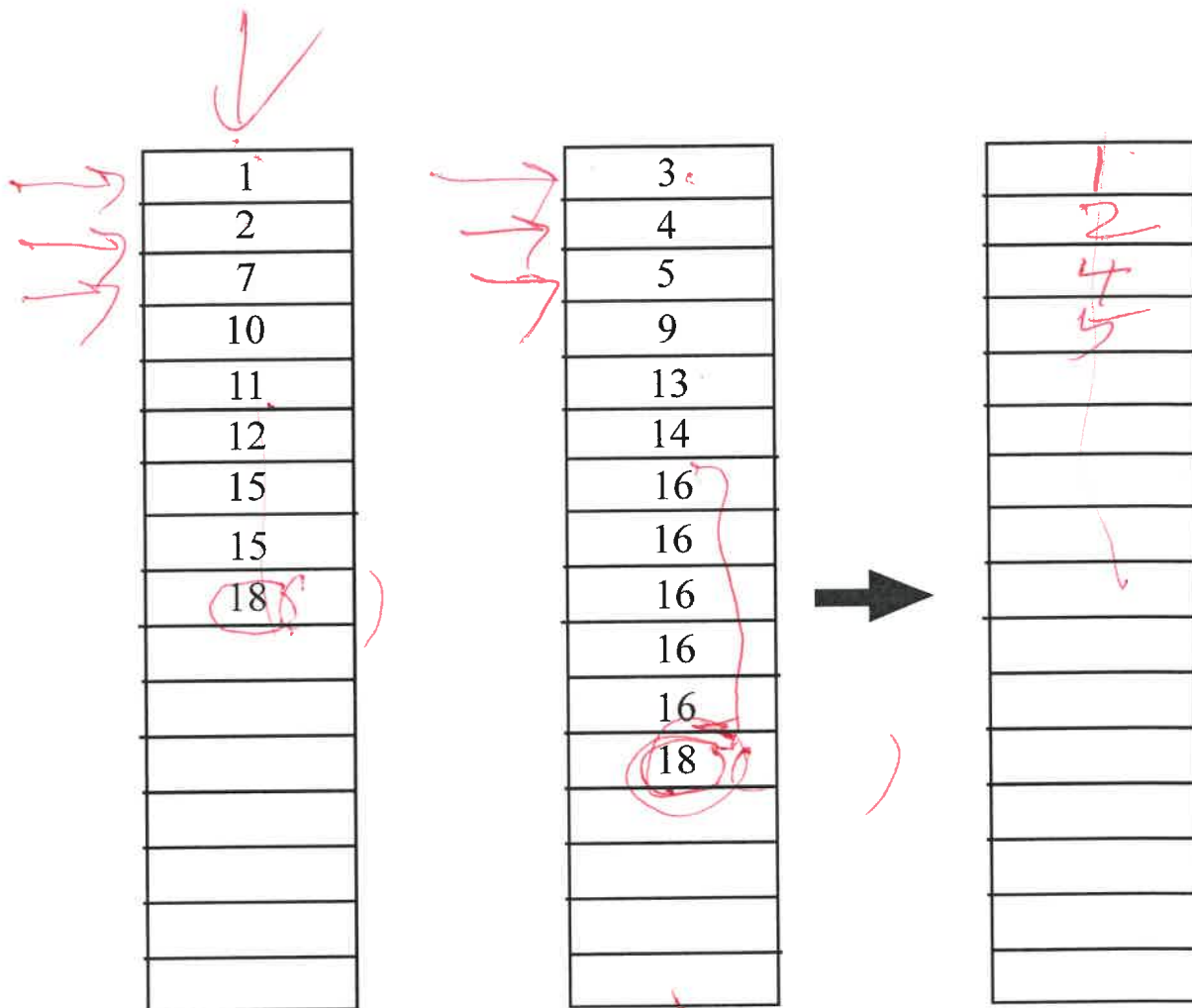
1. Divide into subproblems (unless size allows a trivial solution).
2. Conquer the subproblems.
3. Combine solutions to subproblems.

(Binary) Mergesort – An “Optimal” Key-Comparison Sort (
<https://ranger.uta.edu/~weems/NOTES3318/mergesort.new.c>)

1. Split (copy) array into two sub-arrays (unless $n < 2$).
2. Call Mergesort recursively for each sub-array.
3. Merge together the two ordered sub-arrays.

```
void mymergesort(int *arr, int *work, int n)
{
    int nleft, nright, i;

    if (n < 2)
        return;
    nleft = n/2;
    nright = n - nleft;
    for (i = 0; i < n; i++)
        work[i] = arr[i];
    mymergesort(work, arr, nleft);
    mymergesort(work + nleft, arr + nleft, nright); // pointer arithmetic
    merge(work, work + nleft, arr, nleft, nright); // pointer arithmetic
}
```



```
int merge(int *in1,int *in2,int *out1,int in1Size,int in2Size)
// Merge (union with duplicates) for two ordered tables in1 and in2
// to give out1.
```

```
{
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 items with identical keys ("duplicates") handled?

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

Fall 2009 Test Problem Applying Merge Concept

Two `int` arrays, `A` and `B`, contain `m` and `n` ints each, respectively. The elements within each of these arrays appear in ascending order without duplication (i.e. each table represents a set). Give Java code for a $\Theta(m+n)$ algorithm to find the symmetric difference by producing a third array `C` (in ascending order) with the values that appear in exactly one of `A` and `B` and sets the variable `p` to the final number of elements copied to `C`. (Details of input/output, allocation, declarations, error checking, comments and style are unnecessary.)

```
i=j=p=0;

while (i<m && j<n)
    if (A[i]<B[j])
        C[p++]=A[i++];
    else if (A[i]>B[j])
        C[p++]=B[j++];
    else
    {
        i++;
        j++;
    }

for ( ; i<m; i++)
    C[p++]=A[i];
for ( ; j<n; j++)
    C[p++]=B[j];
```

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

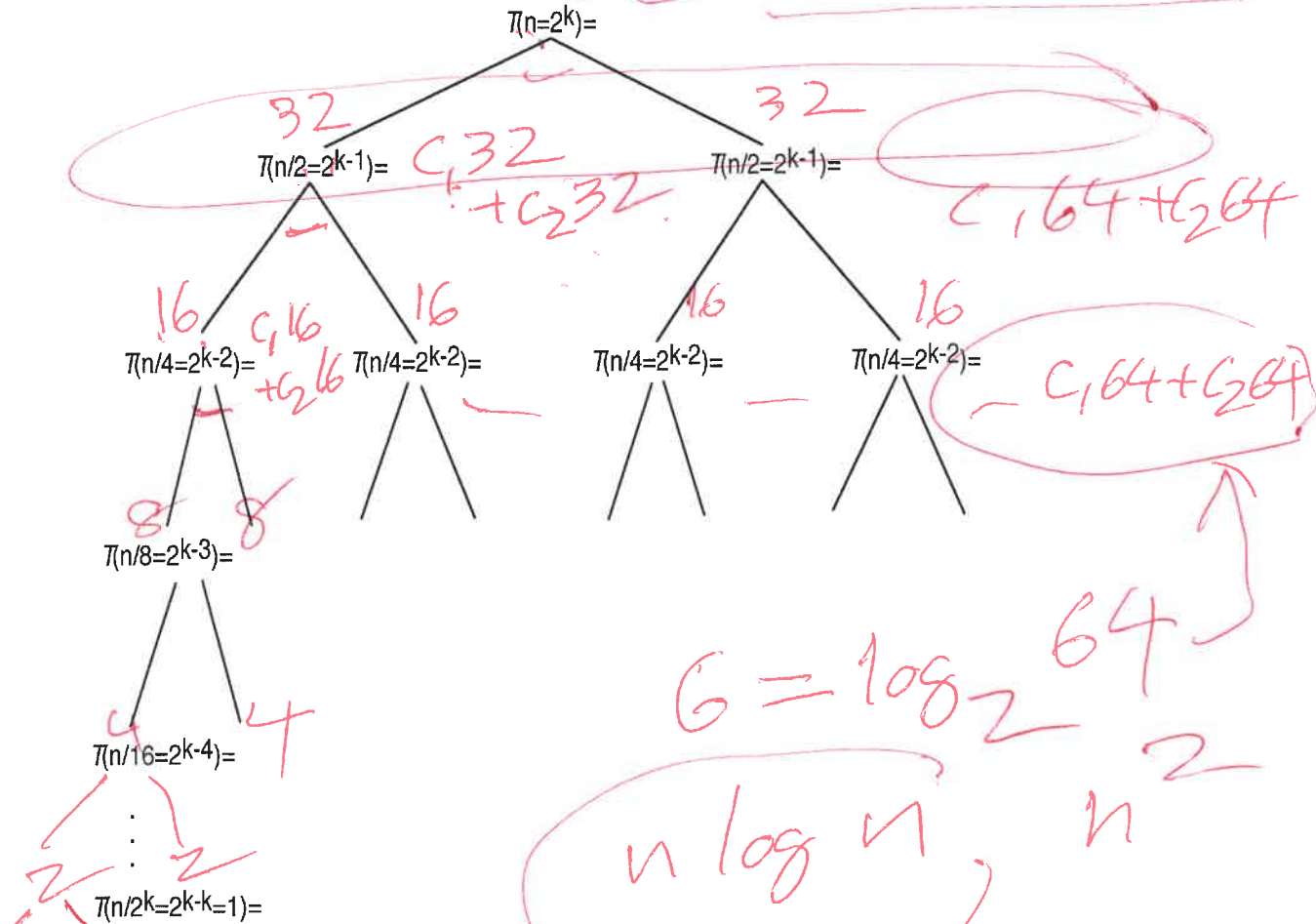
1. Split: n steps. [Can reduce to constant time by pointer arithmetic.]
2. Call recursively:

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

3. Merge together (n steps)

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

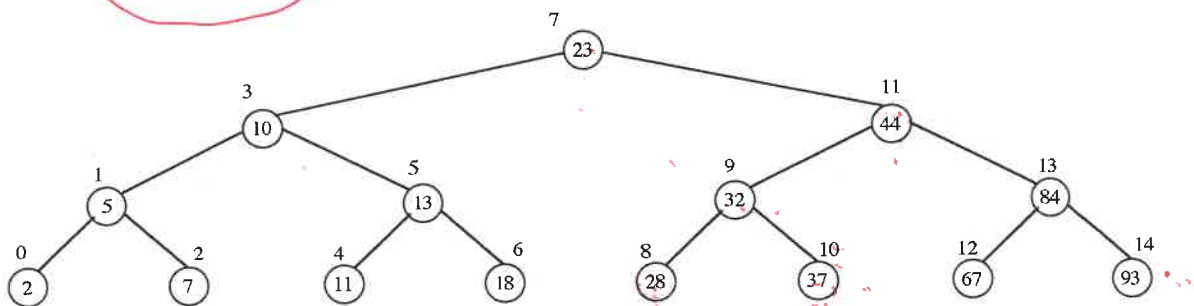
Recursion Tree



[Don't generalize from this example of a recursion tree. More of these in Notes 04.]

1.C. BINARY SEARCH - "Optimal" Search of an Ordered Table (or "Space")

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



(<https://ranger.uta.edu/~weems/NOTES3318/binarySearch.c>)

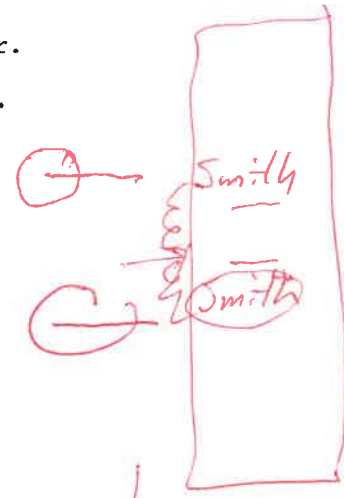
Handwritten notes below the tree:

- 37
- 35
- 10, 9

```

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
}

```



Y Y M M D D H 1 9 5

Recursive binary search?

Multiple occurrences of keys (<https://ranger.uta.edu/~weems/NOTES3318/binarySearchRange.c>)

Find i such that $a[i-1] < \text{key} \leq a[i]$

```

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;
}

```

Relationship of low and high on return?

Find i such that $a[i] \leq \text{key} < a[i+1]$

```

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;
}

```

Relationship of low and high on return?

Partial output from binarySearchRange.c (count is 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