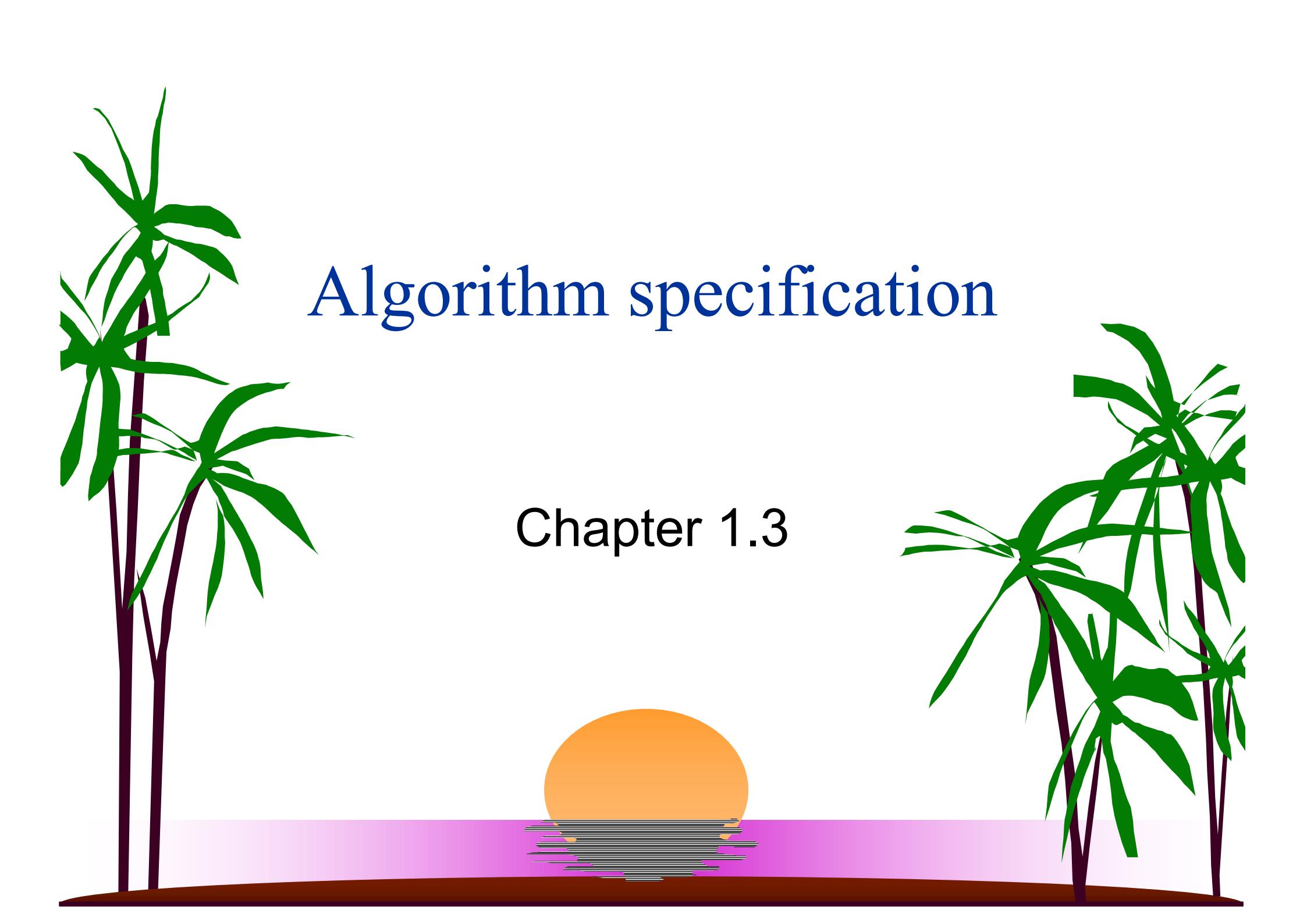


Basic Concepts

Chapter 1

강의 스케줄

1. 1장: Binary search, performance analysis
2. 2장: Arrays and structures
3. 2장: Polynomials, Sparse matrices
4. 3장: Stacks and queues
5. 4장: Linked list
6. 4장: Polynomials, List operations
7. 5장: Binary Trees
8. 5장: Heaps, Binary Search Trees
9. 6장: Graph
10. 6장 : Minimum Cost Spanning Trees
11. 7장: Insertion Sort, Quick Sort
12. 7장: Merge Sort, Heap Sort
13. 8장: Hashing



Algorithm specification

Chapter 1.3

Algorithm specification

Definition An algorithm is a finite set of instructions that, if followed, accomplishes a particular task

All algorithm must satisfy the following criteria:

- 1) Input
- 2) Output
- 3) Definiteness
- 4) Effectiveness

Example: Search problem

Task: Find a specific value n among N distinct numbers

Input. : n , N numbers

Output: 1 if n exists among N numbers
0 otherwise

How?

What kind of data types?

12 5 10
 21 33
7 8

How to search?

12 5 21 10 33 8 7

Millions of numbers

..	2399	5023	3859	588	6545	7611	1951	1615	9683	731	9686	37	4830	2422	709	8648	6772	2531	1325	1651	9546	4024	5820	6274	5643	290	6..
.63	3355	2478	3558	2692	2429	2390	3502	9127	3412	6284	8065	650	5966	4947	1983	5045	1940	817	555	1531	7265	6926	3645	1649	6071	3229	.
.22	7518	7850	617	1374	7606	8989	6449	4028	255	7205	6550	1808	4744	5900	1569	761	8578	4984	494	9983	2374	3792	6064	9871	1904	5889	21..
.77	5842	6466	1095	392	2270	9855	8303	7391	7893	308	1511	9799	5036	220	488	4385	4246	3768	5353	1956	6955	3893	3518	4924	6865	3..	
.16	525	54	9853	2669	9005	9094	271	6224	2071	7751	4615	7266	2711	936	8304	9322	9576	9049	4310	7642	6043	2108	5124	5826	1276	3137	5..
.64	4414	433	7015	7819	666	382	3308	8828	1363	6361	9502	4138	7523	2102	6869	1387	606	3155	585	619	6988	243	4878	629	161	6358	3..
.44	7794	4233	6685	8646	9797	9337	1340	336	4563	3813	5876	8454	1434	1239	3438	384	5213	4872	7148	5722	449	1567	1949	6786	5659	5075	7..
.48	4522	4941	6260	2405	2829	4656	7388	906	1182	6381	5207	9253	866	3720	5330	6654	429	9439	894	810	8230	7099	8471	111	5333	4720	.
.21	5145	9190	7995	4954	7211	9091	1721	4347	8385	2301	7882	6186	2867	1803	9923	5600	5191	3305	1116	2119	6385	7598	5807	8557	2193	1982	21..
.12	5952	3863	5156	8747	4008	9209	9565	9008	6158	251	6264	3353	9594	8924	4749	1831	3959	4453	5251	9554	7223	4351	9368	4238	6179	1624	.
.35	4093	1573	5436	2152	8343	8562	2559	8900	4167	7417	3950	4634	7484	347	2997	2278	620	5735	1490	4168	3240	5147	2276	8249	6378	8055	6..
.82	9669	3994	5964	9481	6235	354	1073	2233	8613	9778	8197	142	9885	3799	8597	8919	1592	6096	7578	8361	8824	1526	1550	7637	729	1524	2..
.53	8649	9973	8507	3529	9851	9984	1552	1953	1305	458	7779	173	7865	9553	273	9948	1704	6647	3166	2315	7704	1058	9593	966	1730	1163	4..
.104	5286	5098	7280	9330	7079	1741	9766	1342	7614	708	6866	4305	7057	1205	159	636	1645	4100	9415	7681	7649	4010	7414	7897	8641	1894	8..
.01	471	7116	7418	6229	4353	4535	4248	4566	8122	2350	8140	2381	420	8031	6998	9954	5689	8589	9401	6369	4237	780	2437	6373	2332	4877	5..
.31	422	3980	7590	9138	9446	4860	8663	2930	6914	4296	4540	9099	7693	5667	2980	6400	5641	3619	2807	2420	8516	5945	136	1821	757	5660	5..
.68	446	8776	3200	4785	1122	4704	1462	6445	4118	4139	5585	7576	1838	2448	2373	1572	2047	9880	5319	991	3641	4536	9041	1015	5588	6293	2..
.43	4458	8049	8263	7714	4450	7869	984	9981	9252	2121	9597	9620	4775	6985	9764	7196	7315	4820	5999	5594	4245	1033	4879	5785	1125	4861	3..
.93	1377	7553	9400	389	6066	5995	5233	3719	8488	7390	8818	4541	6990	1292	5089	7178	3562	4129	8619	3683	466	749	8868	1765	2888	2122	5..
.67	6602	9221	2187	5426	8417	6041	3926	1522	8633	7349	4754	4986	9228	6805	2604	60	7434	3974	1794	2402	896	7837	8437	8581	2303	4975	5..
.134	6456	6818	3064	6516	1066	328	6353	6278	4798	1822	8842	1474	1394	5658	5326	3479	8579	2925	2621	5253	460	4636	7707	8461	8285	5898	2..
.01	9461	1452	5923	6389	1681	4674	9899	3383	4113	8921	2388	5928	2549	48	431	2897	4958	1164	773	742	3164	4060	4662	6175	9312	1400	11..
.56	610	6035	6655	6376	9531	5100	8021	7846	7533	4857	6416	8882	9412	9356	8351	5987	6527	9028	3650	9066	1815	5947	5068	5114	1563	6471	9..
.40	158	2805	6720	3297	1324	3190	2109	1519	6533	5358	9325	3435	4722	196	8373	7272	7663	8515	8887	9961	5788	2288	6323	7191	4602	2875	9..
.42	4271	3384	9055	6819	9410	9684	922	8259	1564	2338	2324	4524	7231	1850	5526	7102	5810	7029	2145	5632	3548	1725	7342	3376	4922	268	3..
.13	6909	8194	1845	9060	9489	5520	1806	3217	140	8224	8830	8935	6758	3039	4461	3286	8161	327	1767	3316	1018	1233	8702	3610	8709	9785	2..
.95	121	68	6001	2014	6472	4135	5413	943	7507	9913	1647	332	8235	9645	7441	2745	9535	5655	427	5462	6567	6139	61	5822	5416	6519	2..
.98	1553	130	7911	9767	9437	7340	6136	3734	9111	5339	43	8129	3128	8993	5513	7098	6399	9654	9260	5869	8840	1382	4254	3004	5599	31..	
.74	6775	3938	7754	4700	1671	350	5610	6213	3729	6256	6546	9230	3755	3348	5331	3228	9727	9214	1051	3434	3924	9024	2683	8841	4933	2132	9..
.20	2339	1158	3197	9807	3583	8064	1202	6723	6658	378	2969	1616	7521	7845	9792	4121	3691	7677	9047	5464	8995	6709	7338	4841	9943	8..	
.138	9734	1460	9872	6872	4395	3108	2059	348	4216	9514	3824	6932	240	2629	8688	6144	6366	6167	7322	189	7756	3424	5095	4207	1054	1353	21..
.36	1976	323	6693	2541	7627	3827	2547	8200	2624	6050	9064	1076	4227	9588	646	2451	841	3948	8155	154	5967	9016	2500	4052	7358	626	21..
.99	7389	5438	8316	3920	5217	2331	6635	8700	2090	712	8119	8398	6835	363	150	6650	7835	3449	7801	3988	8228	1581	2116	6439	8808	4..	
.90	5651	4029	523	6755	5225	807	7378	3719	8980	491	1221	5793	2585	9188	2939	4776	3862	3457	7467	6572	5908	496	3204	605	1370	9157	8..
.08	989	4365	6707	855	3016	1487	4166	6297	338	3322	509	4279	6142	5235	2436	6226	7856	397	9129	7930	9440	8629	6628	9532	2198	1648	7..
.09	2504	7130	331	6746	6890	2242	1559	4949	5627	8240	51	3979	9629	3726	9074	2821	2413	4176	9363	8504	2598	9135	6333	2579	8045	416	9..
.77	2560	5684	7599	2626	804	6831	8421	9403	1628	2973	3346	1356	4051	3609	6941	457	567	4873	1720	2310	9426	4980	4907	7293	5806	2978	9..
.185	1966	3845	6317	8787	2577	9873	736	7980	34987	9796	5344	1822	7263	3587	8179	4088	7022	9787	7999	9338	6132	3266	5349	3579	7539	1..	
.01	3684	9335	7757	9240	8647	2333	1764	8590	7083	546	1706	8690	9726	7131	778	7158	2779	8923	9791	2489	6712	969	3315	1379	9181	8550	5..
.71	2749	6759	199	849	6036	5773	691	2701	102	1243	8907	265	5523	3032	6965	2892	9357	5457	7260	9373	1502	875	6327	2376	3928	163	41..
.02	5392	4693	1716	647	8468	4085	2991	1457	2432	8413	9742	9914	5686	4645	6695	1931	5626	8311	7312	4210	7744	1118	6169	9942	953	9118	2..
.67	1486	5837	699	6253	1890	8078	3432	5751	713	3975	9013	6585	6530	2033	5264	8007	6752	7337	425	688	6843	1957	394	1877	9889	1..	
.72	8657	7628	2076	8774	9564	4611	2490	2280	3783	9441	5752	7643	7728	4748	9958	6829	8362	9097	5374	6204	5						

Better methods?



(1) Sort: Selection sort

- ▲ **Task:** From those integers that are currently unsorted, find the smallest and place it next in the sorted list.
- ▲ **Input:** n integers are stored in an array $\text{list}[i]$, $0 \leq i < n$.
- ▲ **Output:** a sorted list[i], $0 \leq i < n$.
- ▲ **Specification:**

```
for (i=0; i<n; i++) {  
    Examine list[i] to list[n-1] and  
    suppose that the smallest integer is at list[min];  
    Interchange list[i] and list[min];  
}
```

(2) Search: Binary search

▲ Task:

- Assume that we have $n \geq 1$ distinct integers that are already sorted and stored in the array *list*.
- For a given integer m , we want to know whether there is i such that $list[i] = m$.

▲ **Input:** a sorted array $list[i]$, $0 \leq i < n$, an integer m

▲ **Output:** i if $list[i] = m$, -1 if there is no such i

▲ Specification:

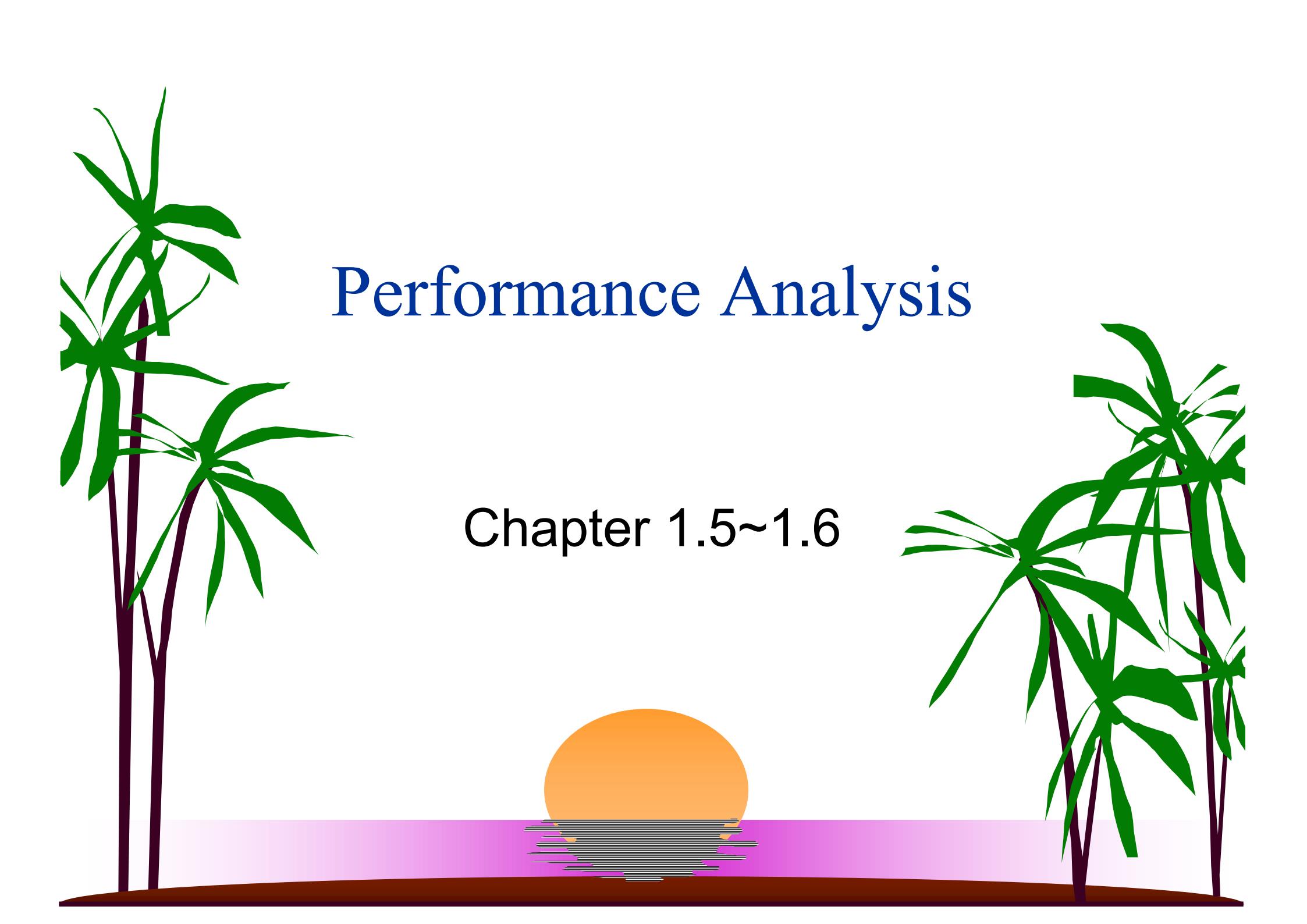
let $left = 0$, $right = n-1$, $middle = (left + right)/2$

- 1) If $m < list[middle]$, then $right = middle - 1$
- 2) If $m = list[middle]$, then return $middle$
- 3) If $m > list[middle]$, then $left = middle + 1$
- 4) Recalculate $middle$ and repeat 1) ~3) while there is more integers to check

Implementation: program structure

```
#define LIST_SIZE 500
int list[LIST_SIZE];
int num_cnt = 0;

void main(int argc, char* argv[]){
    // read data list[] and num_cnt
    ...
    selection_sort(num_cnt);
    // write sorted list
    ...
    fprintf(stdin, "enter the number to search: ");
    fscanf(stdin, "%d", &searchnum);
    result = binsearch(searchnum, 0, num_cnt);
    if(result >=0)
        fprintf(stdout, "number %d is %dth number in the list\n", searchnum, result);
    else fprintf(stdout, "number %d is not in the list\n", searchnum);
    return;
}
```



Performance Analysis

Chapter 1.5~1.6

Performance Analysis

(성능/효율 분석)

- One of the goals of this lecture is to develop your skills for making evaluative judgments about programs.
 - Does the program efficiently use primary and secondary storage?
 - Is the program's running time acceptable for the task?

Which one has better performance?

```
float sum(float list[], int n)
{
    float tempsum = 0;
    int i;
    for (i = 0; i < n; i++)
        tempsum += list[i];
    return tempsum;
}
```

Program 1.11: Iterative function for summing a list of numbers

```
float rsum(float list[], int n)
{
    if (n) return rsum(list, n-1) + list[n-1];
    return 0;
}
```

Program 1.12: Recursive function for summing a list of numbers

Some Uses Of Performance Analysis

- Determine practicality of algorithm
- Predict run time on large instance
- Compare two algorithms that have different asymptotic complexity (점근적 복잡도)
 - e.g., $O(n)$ and $O(n^2)$

Program Complexity

▲ Program complexity

- space complexity : the amount of memory it needs
- time complexity : the amount of computer time it needs

▲ Performance evaluation phases

- performance analysis : a priori estimates (연역적 평가)
- performance measurement : a posteriori testing
(귀납적 테스팅)

Space Complexity

- ▲ Fixed part : independent of the characteristics of the inputs and outputs
 - ▲ Instruction space, space for simple variables, fixed-size structured variables, constants
- ▲ Variable part : depends on the instance characteristics
- ▲ $S(P) = c + Sp(n)$
 - $S(P)$:space requirement of program P
 - c : constant (for fixed space requirements)
 - n : Instance characteristics (ex: I/O size, number)

Space Complexity : Example(1)

```
float abc(float a, float b, float c)
{
    return a+b+b*c+(a+b-c) / (a+b)+4.00;
}
```

Program 1.10: Simple arithmetic function

```
float sum(float list[], int n)
{
    float tempsum = 0;
    int i;
    for (i = 0; i < n; i++)
        tempsum += list[i];
    return tempsum;
}
```

Program 1.11: Iterative function for summing a list of numbers

Space Complexity : Example(2)

```
float rsum(float list[], int n)
{
    if (n) return rsum(list, n-1) + list[n-1];
    return 0;
}
```

Program 1.12: Recursive function for summing a list of numbers

Type	Name	Number of bytes
parameter: array pointer	<i>list[]</i>	4
parameter: integer	<i>n</i>	4
return address: (used internally)		4
TOTAL per recursive call		12

Exercise 1

$n! = 1$, if ($n \leq 1$)
 $n! = n * (n-1)!$, if ($n >= 2$)



```
long factorial(int number)
{
    if (number <= 1)
        return 1;
    else
        return (number * factorial(number - 1));
}
```

Time complexity

$$\blacktriangle T(P) = c + T_p(n)$$

- $T(P)$: time taken by program P
- c : compile time
- T_p : run time
- n : instance characteristics

Program Steps

- ▲ We could count the number of operations the program performs to analyze the time requirements
 - ▲ Machine-independent estimate
 - ▲ We must know how to divide the program into distinct steps
- ▲ **Definition:** *A program step*
 - ▲ A meaningful segment of a program whose execution time is independent of the instance characteristics

Determining the number of steps : examples

Only need to worry about executable statements

```
float sum(float list[], int n)
{
    float tempsum = 0; count++; /* for assignment */
    int i;
    for (i = 0; i < n; i++) {
        count++; /* for the for loop */
        tempsum += list[i]; count++; /* for assignment */
    }
    count++; /* last execution of for */
    count++; /* for return */ return tempsum;
}
```

Program 1.13: Program 1.11 with count statements

Number of steps?

Determining the number of steps : examples

```
float rsum(float list[], int n)
{
    if (n) return rsum(list, n-1) + list[n-1];
    return 0;
}
```

Program 1.12: Recursive function for summing a list of numbers

Number of steps?

Determining the number of steps : examples

```
void add(int a[][][MAX_SIZE], int b[][][MAX_SIZE],
         int c[][][MAX_SIZE], int rows, int cols)
{
    int i, j;
    for (i = 0; i < rows; i++)
        for (j = 0; j < cols; j++)
            c[i][j] = a[i][j] + b[i][j];
}
```

Program 1.16: Matrix addition

Number of steps?

Determining the number of steps : step count table

statement	steps	freq	total steps
float sum (float list[], int n)	0	0	0
{	0	0	0
float tempsum = 0;	1	1	1
int i;	0	0	0
for (i = 0; i < n; i++)	1	n+1	n+1
tempsum += list[i];	1	n	n
return tempsum;	1	1	1
}	0	0	0
total			$2n+3$

Determining the number of steps : step count table

statement	steps	freq	total steps
float rsum(float list[], int n)	0	0	0
{	0	0	0
if (n)	1	n+1	n+1
return rsum(list, n-1) + list[n-1];	1	n	n
return list[0];	1	1	1
}	0	0	0
total			2n+2

Exercise 2

▲ Write the step count table

statement	steps	freq	total steps
void printMatrix(int matrix[] [MAX-SIZE], int rows, int cols) { int i, j; for (i = 0; i < rows; i++) { for (j = 0; j < cols; j++) printf("%d",matrix[i] [j]); printf (" \n"); } }			
total			

Exercise 3

▲ Write the step count table

statement	steps	freq	total steps
void mult(int a[] [MAX-SIZE], int b[] [MAX-SIZE], int c[] [MAX-SIZE]) { int i, j, k; for (i = 0; i < MAX-SIZE; i++) for (j = 0; j < MAX-SIZE; j++) { c[i][j] = 0; for (k = 0; k < MAX-SIZE; k++) c[i][j] += a[i][k]* b[k][j]; } }			
total			

Asymptotic notation (O , Ω , Θ)

- ▲ Determining the exact step count is very difficult
- ▲ The notion of a step is inexact – not so useful for comparative purposes.
 - $100n+10$ vs. $3n+3$
 - $80n$, $85n$, $75n$

We need to be able to make meaningful statements about the time and space complexities of a program.

Definition [Big "oh"]

▲ $f(n)=O(g(n))$ iff there exist positive constants c and n_0 such that $f(n) \leq cg(n)$ for all n , $n \geq n_0$

▲ $g(n)$ is an upper bound

▲ e.g:

$3n+3= O(n)$ as $3n+3 \leq 4n$ for all $n \geq 3$

$3n+3=O(n^2)$ as $3n+3 \leq 3n^2$ for $n \geq 2$

$10n^2 + 4n + 2 =$

$6*2^n + n^2 =$

▲ $O(n)$, $O(n^2)$, $O(n^3)$, $O(n\log n)$

Definition [Omega]

- ▲ $f(n)=\Omega(g(n))$ iff there exist positive constants c and n_0 such that $f(n) \geq cg(n)$ for all $n, n \geq n_0$
- ▲ $g(n)$ is a lower bound
 - e.g: $3n+2=\Omega(n)$
 - $10n^2+4n+2=\Omega(n^2)$
 - $6*2^n + n^2 =$

Definition [Theta]

- ▲ $f(n)=\Theta(g(n))$ iff there exist positive constants c_1, c_2 and n_0 such that
 $c_1g(n) \leq f(n) \leq c_2g(n)$ for all $n, n \geq n_0$

- ▲ $g(n)$ is both an upper and lower bound

e.g: $3n+2=\Theta(n)$

$$10n^2+4n+2=\Theta(n^2)$$

$$6 \times 2^n + n^2 = \Theta(2^n)$$

$$2^*n*m + 2^*n + 1 =$$

Asymptotic Complexity: Examples

Statement	Asymptotic complexity
void add(int a[][]MAX_SIZE] ...)	0
{	0
int i, j;	0
for (i=0; i<rows; i++)	$\Theta(rows)$
for (j = 0; j < cols; j++)	$\Theta(rows.cols)$
c[i][j] = a[i][j] + b[i][j];	$\Theta(rows.cols)$
}	0
Total	$\Theta(rows.cols)$

Practical Complexities

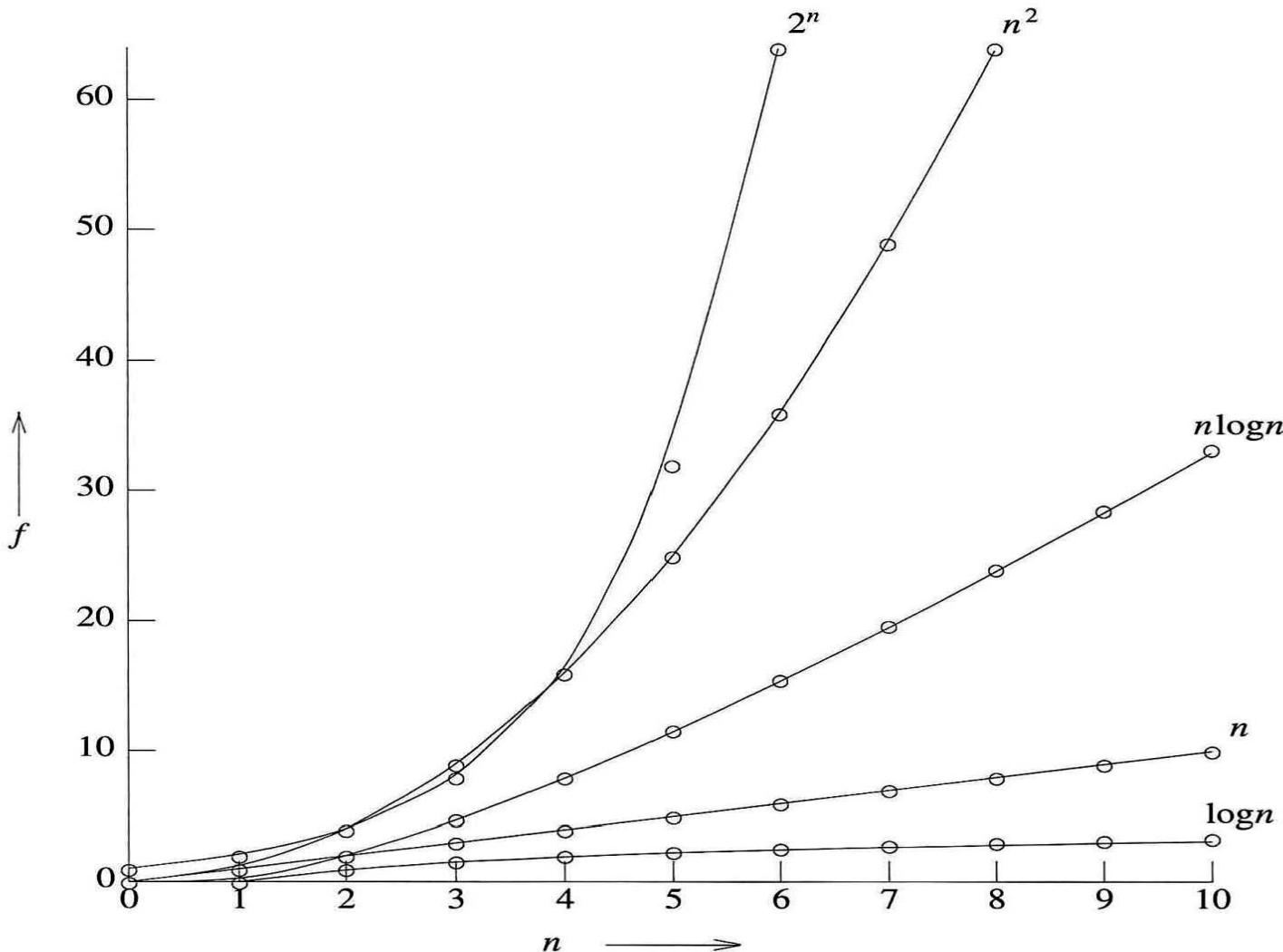


Figure 1.8 Plot of function values

Performance Measurement

- Measure actual time on an actual computer.
- Clocking

Performance Measurement Needs

- Data to use for measurement
 - worst-case data
 - best-case data
 - average-case data
- Timing mechanism --- clock



Event timing In C

- ▲ Use `clock()` function or `time()` function in the C standard library.
- ▲ `#include<time.h>`

	Method 1	Method 2
Start timing	<code>start = clock();</code>	<code>start = time(NULL);</code>
Stop timing	<code>stop = clock();</code>	<code>stop = time(NULL);</code>
Type returned	<code>clock_t</code>	<code>time_t</code>
Result in seconds	<code>duration = ((double) (stop - start)) / CLOCKS_PER_SEC;</code>	<code>duration = (double) difftime(stop, start);</code>

Event timing in C : a sample program

```
#include <stdio.h>
#include <time.h>
#include "selectionSort.h"
#define MAX_SIZE 1001
void main(void)
{
    int i, n, step = 10;
    int a[MAX_SIZE];
    double duration;
    clock_t start;

    /* times for n = 0, 10, ..., 100, 200, ..., 1000 */
    printf("    n      time\n");
    for (n = 0; n <= 1000; n += step)
    {/* get time for size n */

        /* initialize with worst-case data */
        for (i = 0; i < n; i++)
            a[i] = n - i;

        start = clock();
        sort(a, n);
        duration = ((double) (clock() - start))
                    /CLOCKS_PER_SEC;
        printf("%6d    %f\n", n, duration);
        if (n == 100) step = 100;
    }
}
```

n	repetitions	time
0	8690714	0.000000
10	2370915	0.000000
20	604948	0.000002
30	329505	0.000003
40	205605	0.000005
50	145353	0.000007
60	110206	0.000009
70	85037	0.000012
80	65751	0.000015
90	54012	0.000019
100	44058	0.000023
200	12582	0.000079
300	5780	0.000173
400	3344	0.000299
500	2096	0.000477
600	1516	0.000660
700	1106	0.000904
800	852	0.001174
900	681	0.001468
1000	550	0.001818

Program 1.24: First timing program for selection sort

Bad Way To Time

```
do {  
    counter++;  
    start = clock();  
    doSomething();  
    stop = clock();  
    elapsedTime += stop - start;  
} while (elapsedTime < 1000)
```

Event timing in C : more accurate timing

```
#include <stdio.h>
#include <time.h>
#include "selectionSort.h"
#define MAX_SIZE 1001
void main(void)
{
    int i, n, step = 10;
    int a[MAX_SIZE];
    double duration;

    /* times for n = 0, 10, ..., 100, 200, ..., 1000 */
    printf("    n      repetitions      time\n");
    for (n = 0; n <= 1000; n += step)
    {
        /* get time for size n */
        long repetitions = 0;
        clock_t start = clock( );
        do
        {
            repetitions++;

            /* initialize with worst-case data */
            for (i = 0; i < n; i++)
                a[i] = n - i;

            sort(a, n);
        } while (clock( ) - start < 1000);
        /* repeat until enough time has elapsed */

        duration = ((double) (clock() - start))
                    /CLOCKS_PER_SEC;
        duration /= repetitions;
        printf("%6d  %9d  %f\n", n, repetitions, duration);
        if (n == 100) step = 100;
    }
}
```

Program 1.25: More accurate timing program for selection sort

Summary

- ▲ We learned
 - Algorithm specification & basic programming skills
 - ◆ Binary search example
 - Performance analysis
 - ◆ Asymptotic notation
 - Performance measurement
 - ◆ Clocking
- ▲ Those will be used throughout this lecture

Next Time

1. 1장: Binary search, performance analysis
2. 2장: **Arrays and structures**
3. 2장: Polynomials, Sparse matrices
4. 3장: Stacks and queues
5. 4장: Linked list
6. 4장: Polynomials, List operations
7. 5장: Binary Trees
8. 5장: Heaps, Binary Search Trees
9. 6장: Graph
10. 6장 : Minimum Cost Spanning Trees
11. 7장: Insertion Sort, Quick Sort
12. 7장: Merge Sort, Heap Sort
13. 8장: Hashing