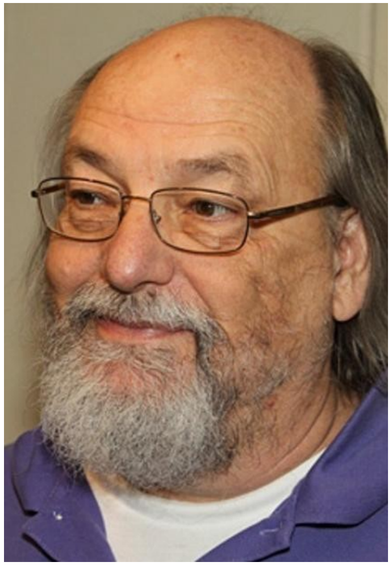
Problem Solving using Arrays

Rupesh Nasre.

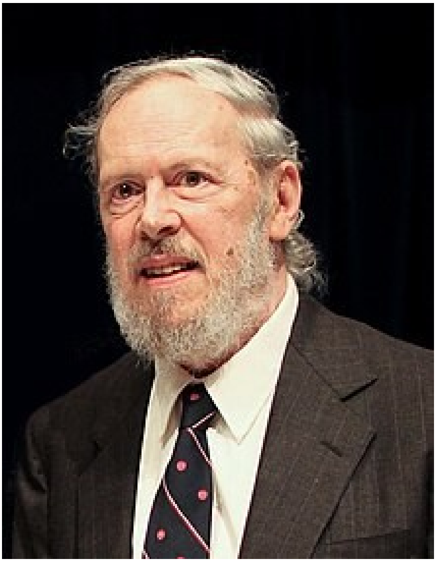
IIT Madras
November 2022

Dennis Ritchie

- Creator of C
- Co-creator of Unix
- Turing Award 1983
  - Nobel Prize in Computing



Ken Thompson



Dennis Ritchie

Photos from wikipedia and NIHF

Week	Problems	Tools
0	Solve equations, find weighted sum.	Data types, expressions, assignments
1	Find max, convert marks to grade.	Conditionals, logical expressions
2	Find weighted sum for all students.	Loops
3	Encrypt and decrypt a secret message.	Character arrays
4	Our first game: Tic-tac-toe	2D arrays
5	Making game modular, reuse.	Functions
6	Find Hemachandra/Fibonacci numbers.	Recursion
7	Encrypt and decrypt many messages.	Dynamic memory, pointers
8	Maintain student records.	Aggregate data types
9	Search and sort student records.	Searching and sorting algorithms
A	Reduce memory wastage.	Linked lists
B	Implement token system in banks.	Queues
C	IRCTC-like ticket booking system	File handling
D	Putting it all together	All the above

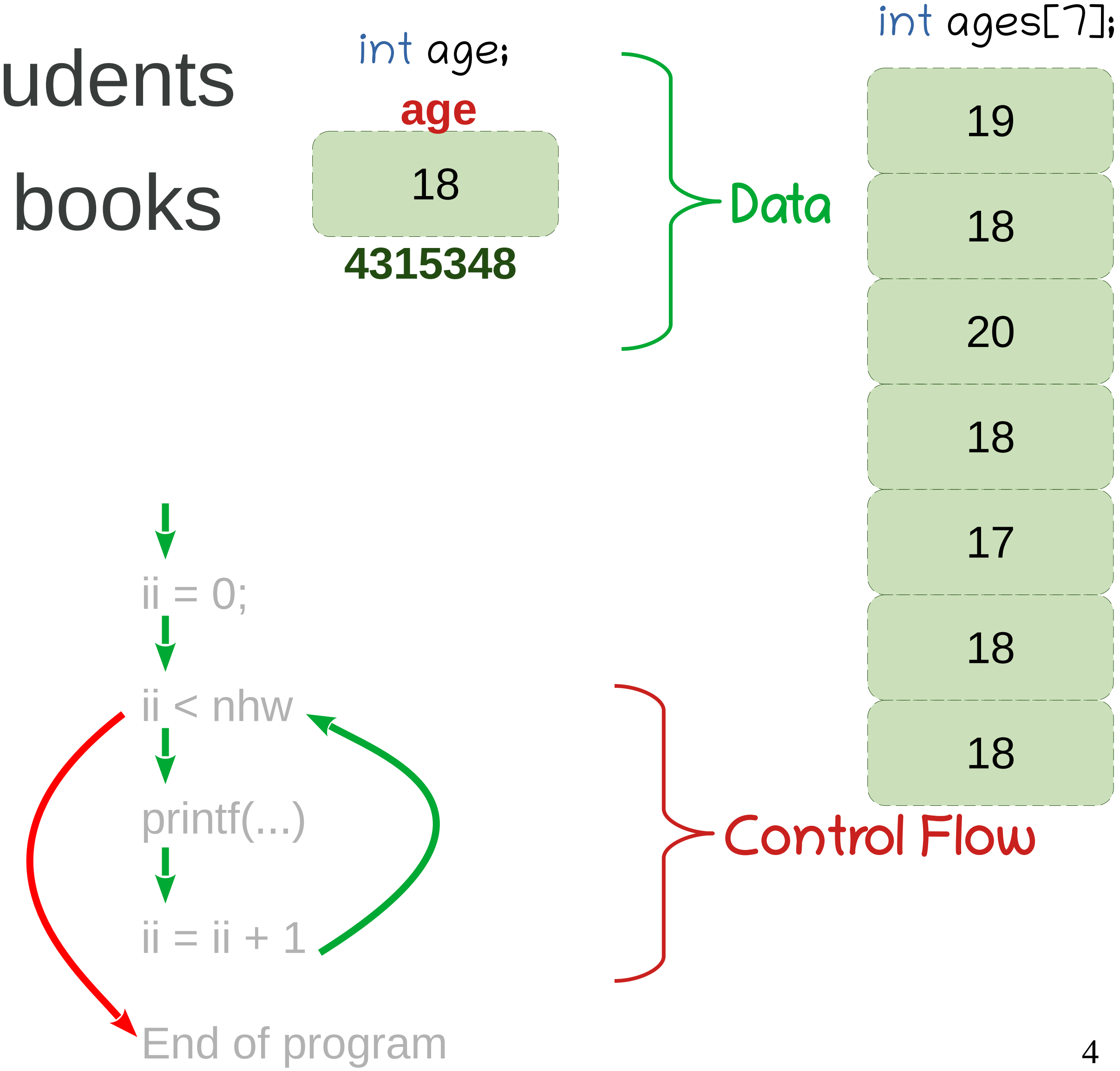
Control Flow

Data

Arrays are uniform aggregates.

- Ages of all the students
- Record of all the books
- A name

- Assignments
- Conditionals
- Loops



int age;

age

18

4315348

Data

int ages[7];

19

18

20

18

17

18

18

ii = 0;

ii < nhw

printf(...)

ii = ii + 1

End of program

Control Flow

Arrays are uniform aggreg

Future Connect:
Different data types can be put together using struct.

```
int main() {
        int ages[7];
        char name[6];

        ages[0] = 19; ages[1] = 18; …
        name[0] = 'A'; name[1] = 'M'; ...

        for (int ii = 0; ii < 6; ++ii)
                printf("%d: %d %c\n",
                        ii, ages[ii], name[ii]);
}
```

int age;

age

4315348	18

int ages[7];

4415320	19
4415324	18
4415328	20
441532C	18
4415330	17
4415334	18
4415338	18

char name[6];

6415396	'A'
6415397	'M'
6415398	'A'
6415399	'N'
641539A	'\0'
641539B	??

Two ways to access a cell:
1. Using the cell's address
2. If I know the start address of the array, then using an offset.

Arrays and loops are friends.

```
#define N 87

int main() {
        int marks[N];
        for (int ii = 0; ii < N; ++ii)
                scanf("%d", &marks[ii]);
        int sum = 0;
        for (int ii = 0; ii < N; ++ii)
                sum += marks[ii];
        printf("Average = %.2f\n", (float)sum / N);
}
```

```
$ ./a.out
87
86
57
93
59
96
...
```

Input redirection

```
$ ./a.out < data.txt
Average = 80.54
```

&marks[0]	4415320	87	marks[0]
&marks[1]	4415324	86	marks[1]
&marks[2]	4415328	57	marks[2]
&marks[3]	441532C	93	marks[3]
⋮	⋮	⋮	⋮

Name of the array is same as the address of its first element.

So, marks == &marks[0]

marks + ii is the address of the ii'th element.

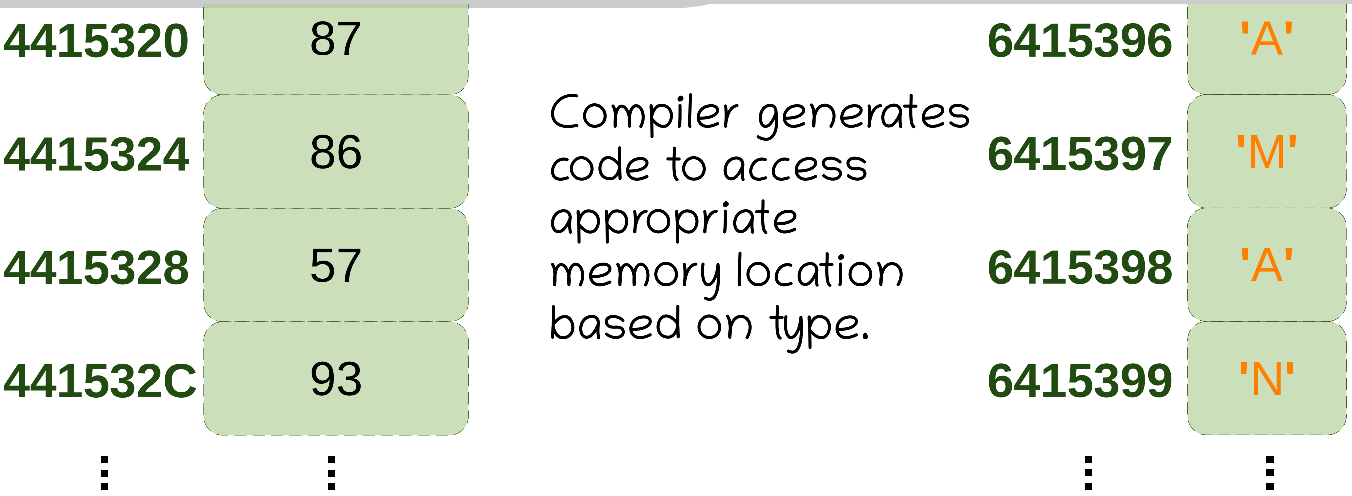
marks + ii == &marks[ii]

Accessing arrays of different types

```
int i, iarray[10];
char c, carray[10];
float f, farray[10];
double d, darray[10];

printf("int=%ld, intarray[10]=%ld\n", sizeof(i), sizeof(iarray));
printf("char=%ld, chararray[10]=%ld\n", sizeof(char), sizeof(carray));
printf("float=%ld, floatarray[10]=%ld\n", sizeof(f), sizeof(farray));
printf("double=%ld, darray[10]=%ld\n", sizeof(d), sizeof(darray));

printf("int addresses: %p %p\n", &iarray[0], &iarray[1]);
printf("char addresses: %p %p\n", &carray[5], &carray[6]);
```

```
int=4, intarray[10]=40
char=1, chararray[10]=10
float=4, floatarray[10]=40
double=8, darray[10]=80
int addresses: 0x7fff3a86a3f0 0x7fff3a86a3f4
char addresses: 0x7fff3a86a453 0x7fff3a86a454
```

bytes



4415320	87
4415324	86
4415328	57
441532C	93
⋮	⋮

Compiler generates code to access appropriate memory location based on type.

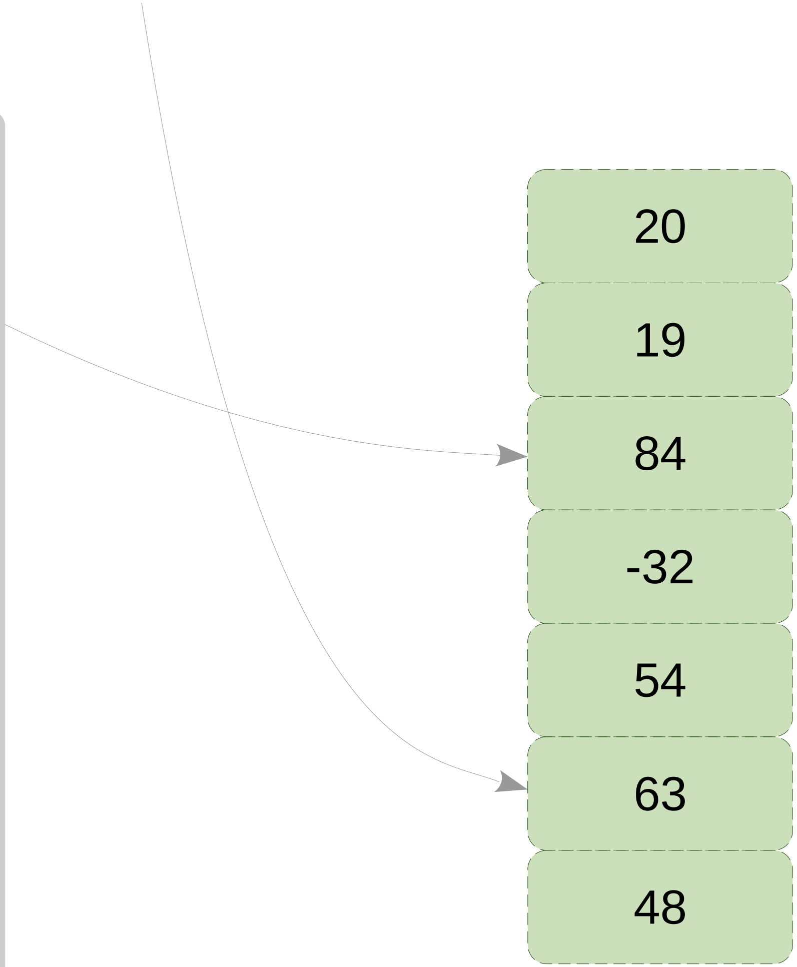
6415396	'A'
6415397	'M'
6415398	'A'
6415399	'N'
⋮	⋮

Find Max and SecondMax

#include <limits.h>

```
// initialize array
int arr[] = {20, 19, 84, -32, 54, 63, 48};
int N = sizeof(arr) / sizeof(arr[0]);

// check for at least two elements
if (N < 2) {
        printf("Add a few more elements.\n");
        exit(1);
}
int max  = INT_MIN;
int smax = INT_MIN;
// find max and smax
for (int ii = 0; ii < N; ++ii) {
        if (arr[ii] > max) {
                smax = max;    ❌
                max = arr[ii];
        }

}
printf("max = %d, smax = %d\n", max, smax);
```

Alternatively, max and smax could be for the first two elements. for loop can start from 2.



Search

```
// initialize array
~~int~~ char arr[] = "Hello World";
int N = sizeof(arr) / sizeof(arr[0]);

// get search key
~~int~~ char key; %c
scanf("~~%d~~", &key);

// search in the array
```

ii is inaccessible.

Is 54 present?
Yes, at index 4.

Is 55 present?
No.

20

19

84

-32

54

63

48

A variable is accessible only within its scope.

Two variables of the same name cannot be defined in the same scope.

But another scope may redefine a variable of the same name.

Alternative approaches:

1. Use a flag to indicate whether the key was found.
2. Remove break to search for all the instances (but be careful).

Let's Play

- One player thinks of a number from 1..1000, and the other player finds it out in 10 questions.
- Only one type of question is allowed
  - Is your number less than some value?
- Two answers are allowed
  - Yes
  - No

Can we always perform a binary search?

Does this problem have a similarity with

- Open page number **273**
- Check the meaning of **supercalifragilisticexpialidocious** in your dictionary
- Tell me the phone number of **Star Garage** from a directory

Binary Search

How is it different from our previous problem of searching in an array?

Linear Search

Binary Search

Write the code.

```
int arr[] = {-5, -3, 0, 4, 43, 58, 59, 64, 70, 74, 75, 79, 81, 88, 92, 93};
int N = sizeof(arr) / sizeof(arr[0]);

int key;
scanf("%d", &key);

int start, end, mid;
start = 0, end = N-1;
while (end - start >= 0) {
        mid = (start + end) / 2;
        if (arr[mid] == key) {
                printf("Found at %d\n", mid);
                break;
        } else if (arr[mid] < key)
                start = mid + 1;
        else end = mid - 1;
        //printf("start = %d, end=%d\n", start, end);
}
if (arr[mid] != key)
        printf("Not present\n");
```

Will it improve performance if I split the array into three parts?

Problem: Reverse the array.

```
 int arr[] = {-5, -3, 0, 4, 43, 58, 59, 64, 70, 74, 75, 79, 81, 88, 92, 93};
 int N = sizeof(arr) / sizeof(arr[0]);

// array reversal




for (int ii = 0; ii < N; ++ii)
        printf("%d ", arr[ii]);
printf("\n");
```

```
#define swap(arr, indexx, indexy)       {  \
        int tmp = arr[indexx];              \
        arr[indexx] = arr[indexy];          \
        arr[indexy] = tmp;                  \
}
```

Problem: Rotate the array.

```
int arr[] = {-5, -3, 0, 4, 43, 58, 59, 64, 70, 74, 75, 79, 81, 88, 92, 93};
int N = sizeof(arr) / sizeof(arr[0]);

// array rotate right




for (int ii = 0; ii < N; ++ii)
        printf("%d ", arr[ii]);
printf("\n");
```

```
// Alternatively
for (int ii = N-1; ii > 0; --ii)
      swap(arr, ii, ii - 1);
```

```
// Further alternatively
int saved = arr[N-1];
for (int ii = 0; ii < N; ++ii) {
      int tmp = arr[ii];
      arr[ii] = saved;
      saved = tmp;
}
```

```
93 -5 -3 0 4 43 58 59 64 70 74 75 79 81 88 92
```

How would you perform a left-rotate?
How to perform a k-rotate?
What do you require to perform a k-rotate in a single loop?

Problem: Duplicates.

```
int arr[N+1];

// read input



// calculate frequencies



// print duplicates
```

```
42 54 3 65 32
54 37 59 37 47
48 42 39 29 22
28 32 51 54 35
21 11 8 23 52
16 35 33 56 65
4 42 55 52
```

```
Duplicate for 32
Duplicate for 35
Duplicate for 37
Duplicate for 42
Duplicate for 52
Duplicate for 54
Duplicate for 65
```

Alternatively,

- For each number, check if it repeats.
- Sort the numbers in ascending order and check consecutive numbers.

Given an attendance list, find the student who has signed more than once (assume roll numbers 1 to 87).

Problem: Negative then Positive.

```
int arr[N] = {53, 33, 0, -4, 43, 9, 58, 22, -59, 4, -7, 74, 55, -9, 23, 8, 2, -3};
```

```
-3 -9 -7 -4 -59 9 58 22 43 4 0 74 55 33 23 8 2 53
```

Given a list of numbers (boys+girls / CS+nonCS / Mahanadi+Ganga / Negative+Positive), move all negatives to the left (in any order).

Problem: Merge sorted arrays

```
int A[] = {-3, 0, 43, 58, 64, 79, 93};
int B[] = {-5, 4, 59, 70, 74, 75, 81, 88, 92};
int NA = sizeof(A) / sizeof(A[0]);
int NB = sizeof(B) / sizeof(B[0]);
```

```
-5 -3 0 4 43 58 59 64 70 74 75 79 81 88 92 93
```

```
int C[NA + NB]; // variable length array, allowed from ANSI C99 standard.
```

```
C[indexC] = A[indexA];
indexA++;
indexC++;
```

Extend the program to perform in-situ merge.
Array A has two sorted sequences.

C = A merge B, with A and B are sorted.
C is also sorted.

char array

```
char arr[] = "Hello";
char two[] = " World";
```

arr	{H', 'e', 'l', 'l', 'o'}	"Hello"	"Bye"	""
N	5	6	4	1

Strings end with ascii value zero.

This can be exploited in various string related functions:

- computing length of a string
- converting a string to upper case
- concatenating two strings

Write these codes.

\0 is an escape sequence.

Escape Sequences

```
int main() {
        printf("Hello\rCarriage Return\n");
        printf("Backspace\b: Check\n");
        printf("Alert\a: Do you hear it?\n");
        printf("Formfeed\fUseful for printing\n");
        printf("T\ta\tb\n");
        printf("Vertical\vTab\n");
        printf("Back\\slash\n");
        printf("%cSingle quote%c\n", '\'', '\'');
        printf("\"Double quote\"\n");
        printf("\x45\x73\x63\x61\x70\x65 Character\n");
        printf("\117\143\164\141\154\n");
        printf("Unicode \u20B9\n");
}
```

```
Carriage Return
Backspac: Check
Alert: Do you hear it?
Formfeed
        Useful for printing
T       a       b
Vertical
        Tab
Back\slash
'Single quote'
"Double quote"
Escape Character
Octal
Unicode ₹
```

System of Equations

$4x – y - 5z = 7$

$2x + 3y - 4z = 9$

$x + z = 22$

$$\begin{pmatrix} 4 & -1 & -5 \\ 2 & 3 & -4 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 7 \\ 9 \\ 22 \end{pmatrix}$$

Can be represented using 1D arrays or 2D array.

Can be represented using 1D arrays.

2D arrays allow us to access an element using its row and column directly (e.g., mat[row][col]).

Useful in image processing, tabular data, etc.

2D Arrays / Matrices

1D Array

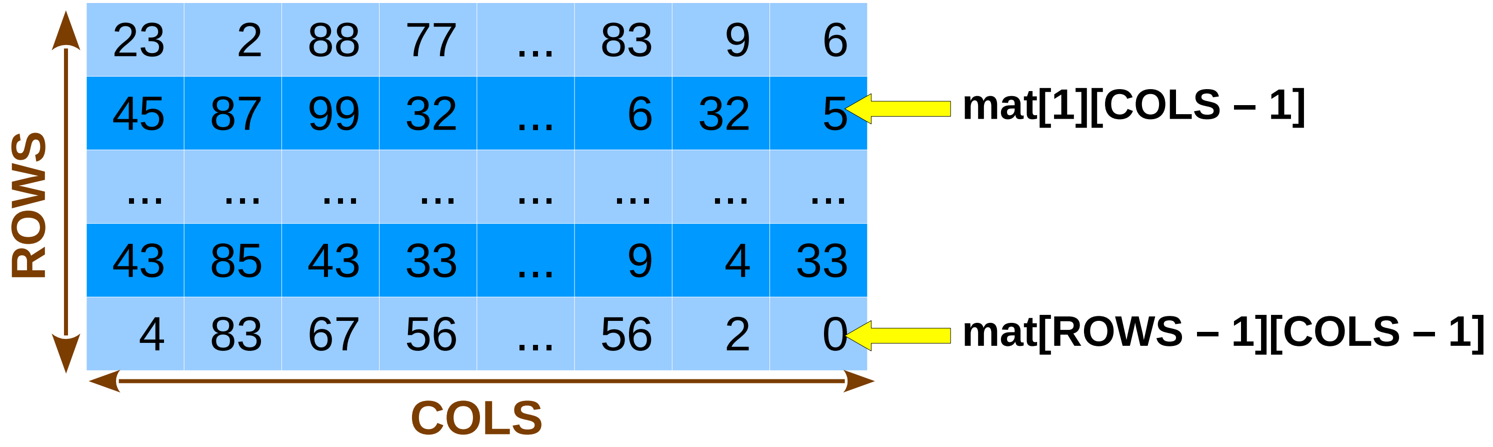
#include <stdlib.h>

```
int mat[ROWS][COLS];

for (int ii = 0; ii < ROWS; ++ii)
        for (int jj = 0; jj < COLS; ++jj)
                mat[ii][jj] = rand() % 100;

for (int ii = 0; ii < ROWS; ++ii) {
        for (int jj = 0; jj < COLS; ++jj)
                printf("%4d", mat[ii][jj]);
        printf("\n");
}
```

Declaration / Definition

Initialization

Printing



23	2	88	77	...	83	9	6
45	87	99	32	...	6	32	5
...	...	...	...	...	...	...	...
43	85	43	33	...	9	4	33
4	83	67	56	...	56	2	0

int mat[ROWS * COLS];

mat[ii S + jj] = ...

23
2
...
6
45
87
...
5
...
4
33
4
83
...
0

Problem: Saddle Point

- A saddle point is minimum in its row and maximum in its column (used in minimax method).

8	11	23	6	5
33	0	9	-3	3
20	-4	6	5	5
12	8	9	7	10

Does it always exist?

Can there be multiple saddle points?

Write the code.

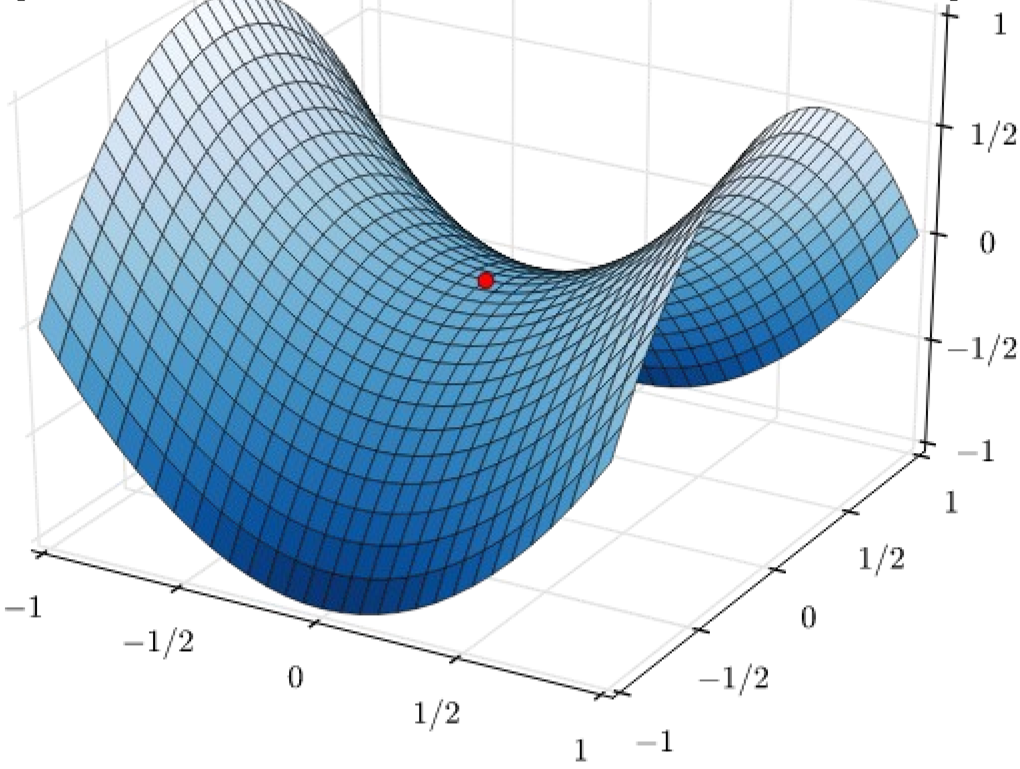


Image: wikipedia

Sorting

- A fundamental operation
- Elements need to be stored in increasing order.
  - Some methods would work with duplicates.
  - Algorithms that maintain relative order of duplicates from input to output are called stable.
- Comparison-based methods
  - Insertion, Bubble, Selection, Shell, Quick, Merge
- Other methods
  - Radix, Bucket, Counting

Sorting Algorithms at a Glance

Algorithm	**Worst case complexity**	**Average case complexity**
Bubble	$O(n^2)$	$O(n^2)$
Insertion	$O(n^2)$	$O(n^2)$
Shell	$O(n^2)$	Depends on increment sequence
Selection	$O(n^2)$	$O(n^2)$
Heap	$O(n \log n)$	$O(n \log n)$
Quick	$O(n^2)$	$O(n \log n)$ depending on partitioning
Merge	$O(n \log n)$	$O(n \log n)$
Bucket	$O(n\ \alpha \log \alpha)$	Depends on α

Bubble Sort

- Compare **adjacent** values and swap, if required.
- How many times do we need to do it?
- What is the invariant?
  - After ith iteration, i largest numbers are at their final places.
  - An element may move *away* from its final position in the intermediate stages (e.g., check the 2nd element of a reverse-sorted array).
- **Best** case: Sorted sequence
- **Worst** case: Reverse sorted (n-1 + n-2 + ... + 1 + 0)
- **Classwork**: Write the code.

Bubble Sort

```
for (ii = 0; ii < N; ++ii)
    for (jj = 0; jj < N - 1; ++jj)
        if (arr[jj] > arr[jj + 1]) swap(jj, jj + 1);
```

Not using ii

```
for (ii = 0; ii < N - 1; ++ii)
    for (jj = 0; jj < N – ii - 1; ++jj)
        if (arr[jj] > arr[jj + 1]) swap(jj, jj + 1);
```

$O(n^2)$

- **Best** case: Sorted sequence
- **Worst** case: Reverse sorted (n-1 + n-2 + ... + 1 + 0)
- What do we measure?
  - Number of comparisons
  - Number of swaps (bounded by comparisons)
- Number of comparisons remains the same!

Insertion Sort

- Consider i^{th} element and insert it at its place w.r.t. the first i elements.
  - Resembles insertion of a playing card.
- Invariant: Keep the first i elements sorted.
- **Note:** Insertion is in a sorted array.
- Complexity: O(n log n)?
  - Yes, binary search is O(log n).

    But are we doing more work?
  - Best case, Worst case?
- **Classwork**: Write the code.

Insertion Sort

```
for (ii = 1 ; ii < N; ++ii) {
        int key = arr[ii];
        int jj = ii - 1;

        while (jj >= 0 && key < arr[jj]) {
                arr[jj + 1] = arr[jj];
                --jj;
        }
        arr[jj + 1] = key;
}
```

i^{th} element

Shift elements
0 + 1 + 2 + ... n-1

At its place

- **Best** case: Sorted: while loop is O(1)
- **Worst** case: Reverse sorted: $O(n^2)$

Selection Sort

- Approach: Choose the minimum element, and push it to its final place.
- What is the invariant?
    - First i elements are at their final places after i iterations.
- **Classwork:**

```
for (ii = 0 ; ii < N - 1; ++ii) {
        int iimin = ii;

        for (jj = ii + 1; jj < N; ++jj)
                if (arr[jj] < arr[iimin])
                        iimin = jj;
        swap(iimin, ii);
}
```

Find min.

Shell Sort

- The number of shiftings is too high in insertion sort. This leads to high inefficiency.
- Can we allow some perturbations initially and fix them later?
- **Approach**: Instead of comparing adjacent elements, compare those that are some distance apart.
  - And then reduce the distance.

Input	81	94	11	96	12	35	17	95	28	58	41	75	15
gap=5	35	17	11	28	12	41	75	15	96	58	81	94	95
gap=3	28	12	11	35	15	41	58	17	94	75	81	96	95
gap=1	11	12	15	17	28	35	41	58	75	81	94	95	96

Shell Sort

```
for (gap = N/2; gap; gap /= 2)
    for (ii = ... ; ii < N; ++ii) {
            int key = arr[ii];
            int jj = ii ~~- 1~~;

            while (jj - gap >= 0 && key < arr[jj - gap]) {
                    arr[jj + 1] = arr[jj];
                    jj -= gap;
            }
            arr[jj~~ + 1~~] = key;
    }
```

i^{th} element

Shift elements

At its place

- **Best** case: Sorted: while loop is O(1)
- **Worst** case: $O(n^2)$

Heapsort

Given N elements,

build a heap and

then perform N deleteMax,

store each element into an array.

N storage

O(N) time

O(N log N) time

O(N) time and N space

O(N log N) time and 2N space

```
        for (int ii = 0; ii < nelements; ++ii) {
                h.hide_back(h.deleteMax());
        }
        h.printArray(nelements);
```

Can we avoid the second array?

Quicksort

- Approach:
  - Choose an arbitrary element (called pivot).
  - Place the pivot at its final place.
  - Make sure all the elements smaller than the pivot are to the left of it, and ... (called partitioning)
  - Divide-and-conquer.

```
void quick(int start, int end) {
        if (start < end) {
                int iipivot = partition(start, end);
                quick(start, iipivot - 1);
                quick(iipivot + 1, end);
        }
}
```

Crucially decides the complexity.

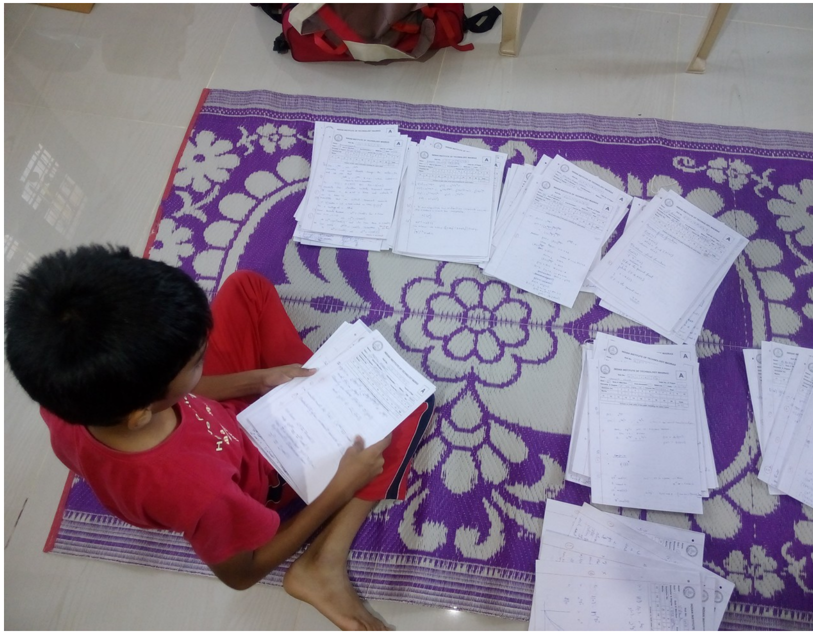
Merge Sort

- Divide-and-Conquer
  - Divide the array into two halves
  - Sort each array separately
  - Merge the two sorted sequences
- Worst case complexity: O(n log n)
  - Not efficient in practice due to array copying.
- **Classwork:**

```
void mergeSort(int start, int end) {
        if (start < end) {
                int mid = (start + end) / 2;
                mergeSort(start, mid);
                mergeSort(mid + 1, end);
                merge(start, mid, end);
        }
}
```

Bucket Sort

- Hash / index each element into a bucket.
- Sort each bucket.
  - use other sorting algorithms such as insertion sort.
- Output buckets in increasing order.
- Special case when number of buckets >= maximum element value.
- Unsuitable for arbitrary types.



Counting Sort

- Bucketize elements.
- Find count of elements in each bucket.
- Perform prefix sum.
- Copy elements from buckets to original array.

Original array	4	1	4	9	11	7	8	1	3	4
Buckets	1, 1		3	4, 4, 4	7		8		9	11
Bucket sizes	2	0	1	3	1	0	1	0	1	1
Starting index	0	2	2	3	6	7	7	8	8	9
Output array	1	1	3	4	4	4	7	8	9	11

Radix Sort

O(P * (N + B))
P = passes
N = elements
B = buckets

- Generalization of bucket sort.
- Radix sort sorts using different digits.
- At every step, elements are moved to buckets based on their ith digits, starting from the least significant digit.
- **Classwork**: 33, 453, 124, 225, 1023, 432, 2232

64	8	216	512	27	729	0	1	343	125
0	1	512	343	64	125	216	27	8	729
00, 01, 08	512, 216	125, 27, 729		343		64			
000, 001, 008, 027, 064	125	216	343		512		729		

String Functions

```
char arr[] = "Hello";
char two[] = " World";

int ii;
for (ii = 0; arr[ii] != '\0'; ++ii)
        ;
int arrlen = ii;
printf("Length of %s is %d\n", arr, arrlen);

for (ii = 0; arr[ii] != '\0'; ++ii)
        if (arr[ii] >= 'a' && arr[ii] <= 'z')
               arr[ii] += 'A' - 'a';
printf("Uppercase %s\n", arr);

char cat[100];
for (ii = 0; arr[ii] != '\0'; ++ii)
        cat[ii] = arr[ii];
for (int jj = 0; two[jj] != '\0'; ++ii, ++jj)
        cat[ii] = two[jj];
cat[ii] = '\0';
printf("%s + %s is %s\n", arr, two, cat);
```

```
char arr[] = "Hello";
char two[] = " World";

int arrlen = strlen(arr);
printf("Length of %s is %d\n", arr, arrlen);

for (ii = 0; arr[ii] != '\0'; ++ii)
       arr[ii] = toupper(arr[ii]);      // needs ctype.h
printf("Uppercase %s\n", arr);

char cat[strlen(arr) + strlen(two) + 1];
strcpy(cat, arr);
strcat(cat, two);
printf("%s + %s is %s\n", arr, two, cat);
```

When you use these functions,
remember that you are incurring a
performance penalty.

Problem: Encrypt / Decrypt Message

```
if (input == 'e') inc = 1;
else if (input == 'd') inc = -1;
...

char c;
while ((c = getchar()) != '\n') {
        c += inc;
        putchar(c);

 }
```

```
if (input == 'e') inc = 1;
else if (input == 'd') inc = -1;

char s[100];
gets(s);

int ii = 0;
while (s[ii] != '\0') {
        s[ii] += inc;
        ++ii;
 }
 puts(s);
```

Problem: Find if you are ~~rusticated~~ awarded.

- Given a long string containing names of students ~~rusticated~~ awarded, find out if you are in it.

The W,

ARJU nan.

```
gets(message);        // works with input string having spaces.
char key[] = "THARUN DYANISH";
enum {SAME, DIFFERENT} comparison = DIFFERENT;

if (comparison == SAME)
        printf("The student %s is awarded.\n", key);
else
        printf("No action is required for %s\n", key);
```

When you access array[index], check if index is < array length.

Extend it for
- case insensitive comparison
- allowing arbitrary spaces
- similar sounding words “Tarun Danish“

```
if (strstr(message, key))
        printf(“Awarded“);
else printf(“No action“);
```

	Week	Problems	Tools
✓	0	Solve equations, find weighted sum.	Data types, expressions, assignments
✓	1	Find max, convert marks to grade.	Conditionals, logical expressions
✓	2	Find weighted sum for all students.	Loops
✓	3	Encrypt and decrypt a secret message.	Character arrays
	4	Our first game: Tic-tac-toe	✓2D arrays
	5	Making game modular, reuse.	Functions
	6	Find Hemachandra/Fibonacci numbers.	Recursion
	7	Encrypt and decrypt many messages.	Dynamic memory, pointers
	8	Maintain student records.	Aggregate data types
	9	Search and sort student records.	Searching and sorting algorithms
	A	Reduce memory wastage.	Linked lists
	B	Implement token system in banks.	Queues
	C	IRCTC-like ticket booking system	File handling
	D	Putting it all together	All the above