Arrays

Rupesh Nasre.
rupesh@cse.iitm.ac.in

August 2021

Properties

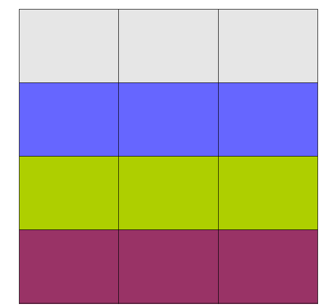
- Simplest data structure
  - Acts as aggregate over primitives or other aggregates
  - May have multiple dimensions
- Contiguous storage
- Random access in O(1)
- Languages such as C use type system to index appropriately
  - e.g., a[i] and a[i + 1] refer to locations based on type
- Storage space:
  - Fixed for arrays
  - Dynamically allocatable but fixed on stack and heap
  - Variable for vectors (internally, reallocation and copying)

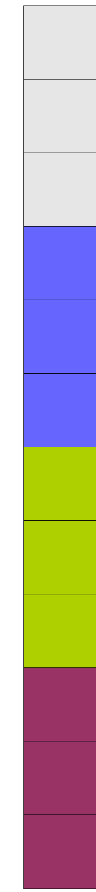
Array Expressions

```
void fun(int a[ ][ ]) {
        a[0][0] = 20;
}
void main() {
        int a[5][10];
        fun(a);
        printf("%d\n", a[0][0]);
}
```

ERROR: **type of formal parameter 1 is incomplete**

We view an array to be a D-dimensional matrix. However, for the hardware, it is simply single dimensional.



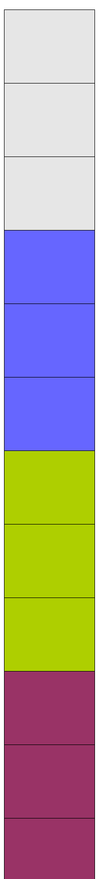


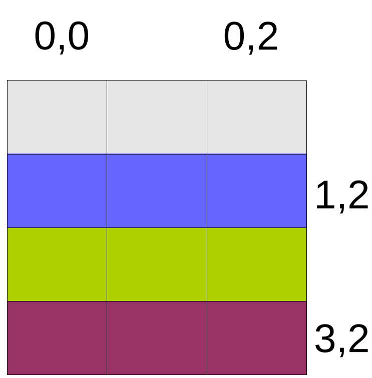
For declaration int a[w4][w3][w2][w1]:

- What is the address of a[i][j][k][l]?
  - (i * w3 * w2 * w1 + j * w2 * w1 + k * w1 + l) * 4
- How to optimize the computation?
  - Use **Horner's rule**: (((i * w3 + j) * w2 + k) * w1 + l) * 4

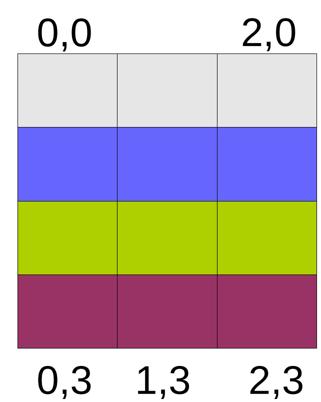
Array Expressions

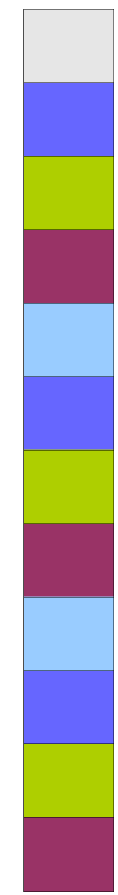
- In C, C++, Java, we use *row-major* storage.
  - All elements of a row are stored together.





- In Fortran, we use *column-major* storage.
  - each column is stored together.

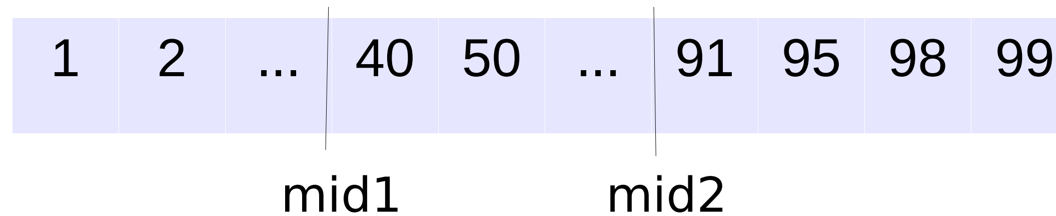




Search

- Linear: O(N)
- Binary: O(log N)
  - $T(N) = T(N/2) + c$

How about Ternary search?




```
int bsearch(int a[], int N, int val) {
        int low = 0, high = N - 1;

        while (low <= high) {
                int mid = (low + high) / 2;
                if (a[mid] == val) return 1;
                if (a[mid] > val) high = mid - 1;
                else low = mid + 1;
        }
        return 0;
}
```

Matrices

- Typically 2D arrays
  - Sometimes array of arrays (int *arr[N])
- If a matrix is sorted left-to-right and top-to-bottom, can we apply binary search?
- Knight's tour
  - Start from a corner.
  - Visit all 64 squares without visiting a square twice.
  - The only moves allowed are 2.5 places.
  - Cannot wrap-around the board.

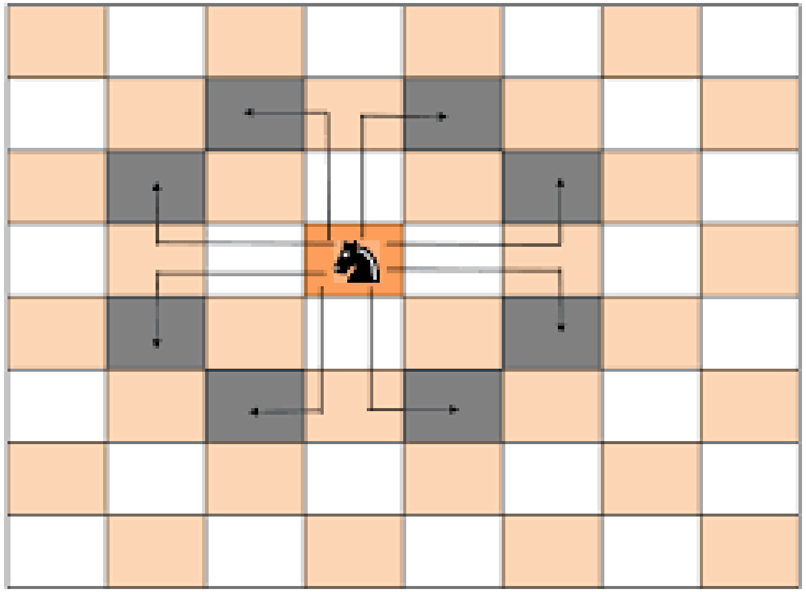


Image source: tutorialhorizon.com

Search in a Sorted Matrix[M][N]

3	5	9	20	39
4	6	11	21	40
7	10	12	23	45
8	13	22	27	46
19	29	41	43	49
24	30	**44**	50	52
25	31	47	51	55
28	33	48	53	61
32	42	54	56	66
35	57	60	62	69

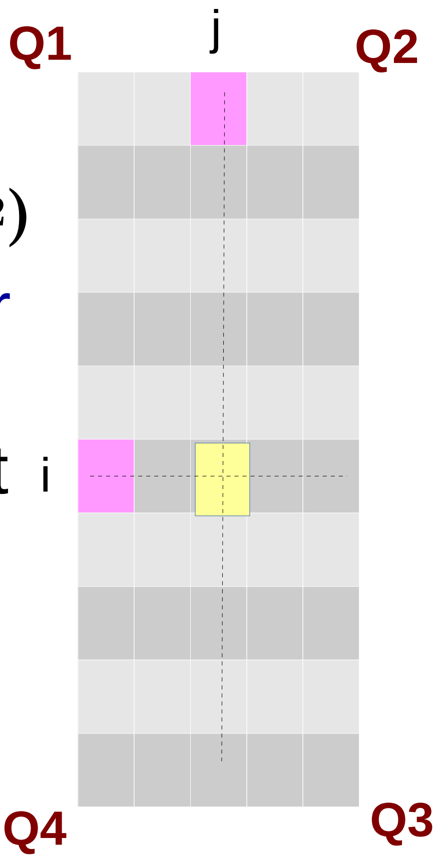
Focus on **44**.
Check where all values < 44 appear.
Check where all values > 44 appear.

Classwork: Devise a method to search for an element in this matrix.

For now, let's assume that all values are unique.

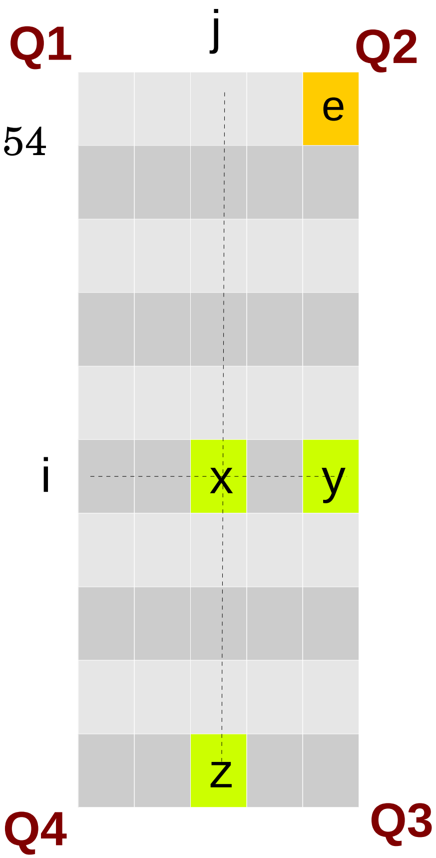
Search in a Sorted Matrix[M][N]

- Approach 1: Divide and Conquer
  - < i, 0 and < 0, j → Q1
  - < i, 0 and > 0, j → Q1, Q2
  - > i, 0 and < 0, j → Q1, Q4
  - > i, 0 and > 0, j → Q1, Q2, Q3, Q4
  - $T(M, N) = 4T(M/2, N/2) + c = O(\min(M, N)^2)$
  - This complexity is same as that for the linear search.
  - To improve complexity, we need to reduce at least one quadrant.
  - Note: A number in Q1 is always smaller than [i,j]. But a number smaller than [i, j] need not be in Q1.



Search in a Sorted Matrix[M][N]

- Approach 2: Divide and Conquer
  - Use the corner points of Q1, Q2, Q3, Q4 to decide the quadrant.
  - > y and > z → Q3
  - Else → Q1, Q2, Q4
  - $T(M, N) = 3T(M/2, N/2) + c = O(\min(M, N))^{1.54}$
- Approach 3: Elimination
  - Consider e: [0, N-1].
  - If key == e, found the element
  - If key < e, eliminate that column
  - If key > e, eliminate that row
  - $O(M + N)$
  - What other corner points I can start with?



Surprise Quiz

- What is *Triskaidekaphobia*?
- What is *Paraskevidekatriaphobia*?

Stall numbers at Santa Anita Park progress from 12 to 12A to 14.

Numbers in a lift

Source: wikipedia

Arrays: Classwork

- Merge two sorted arrays
  - In a third array
  - *In situ* (also check with linked lists)
- For a given data, create a histogram
  - Numbers of students in [0..10), [10, 20), ..., [90, 100].
- Given two arrays of sizes N1 and N2, find a product matrix (P[i][j] = A[i] * B[j]).
  - Can this be done in O(N1 + N2) time?
  - or O(N1 log N2)?

Classwork

- Given an unsorted array of roll numbers, find the smallest CS18 roll number absent today.
  - {2, 3, 7, 6, 8, CH..., 10, 15} outputs 1
  - {2, 3, EE..., 6, 8, 1, CH..., 15} outputs 4
  - {1, 1, EE..., EE..., EE...} outputs 2
- Can this be done in linear time and constant additional space?

Source: geeksforgeeks.com

8-Queens Problem

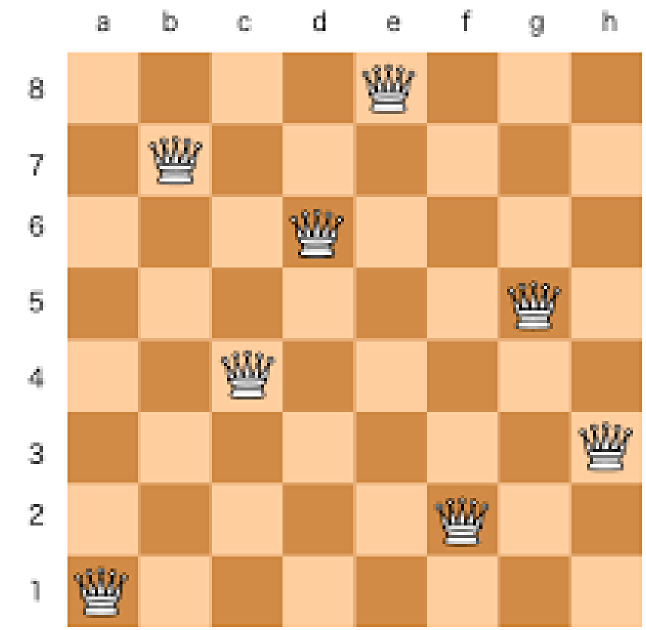
Given a chess-board,

can you place 8 queens

in non-attacking positions?

(no two queens in the same row

or same column or same diagonal)



- Does a solution exist for 2x2, 3x3, 4x4?
- Have you seen similar constraints somewhere?

Image source: leetcode.com

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, Shell, Selection, 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 i^{th} 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 2^{nd} 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: $O(n^2)$

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.
  - This sequence of distances is called increment sequence.

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: $O(n^2)$

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.

Heapsort

Given N elements, — N storage

build a heap and — O(N) time

then perform N deleteMax, — O(N log N) time

store each element into an array. — 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);
```

Source: heap-sort.cpp

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.

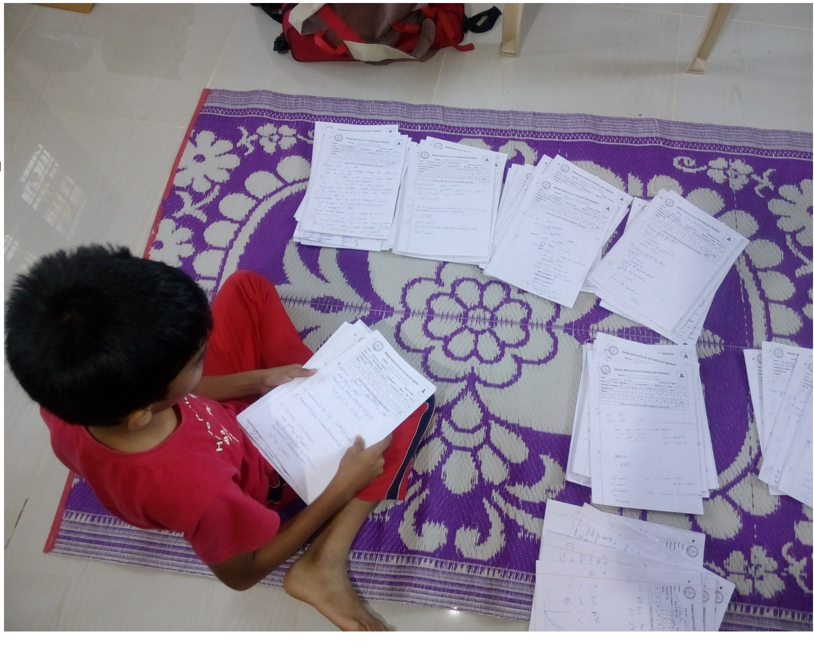
Merge Sort

- 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

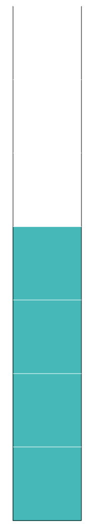
O(P * (N + B))
P = passes
N = elements
B = buckets

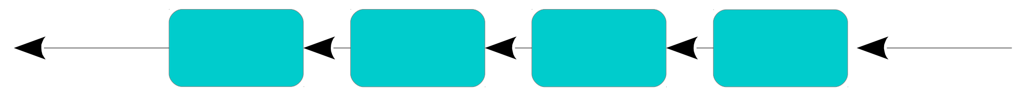
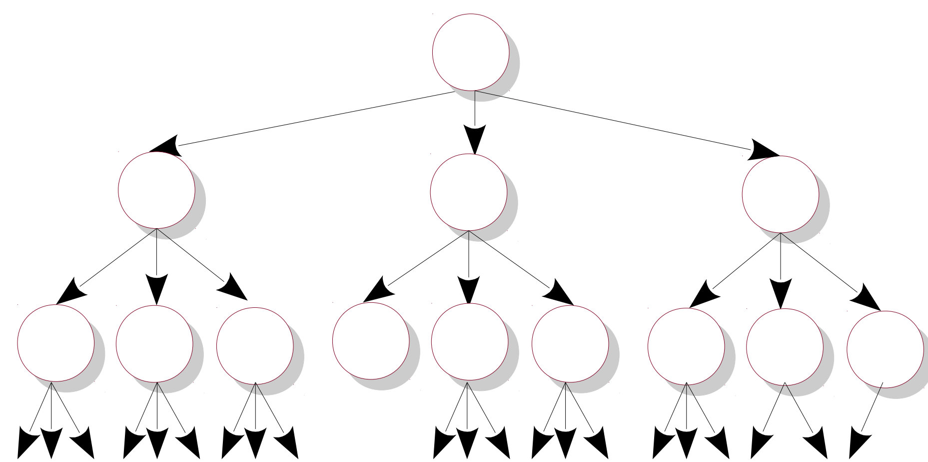
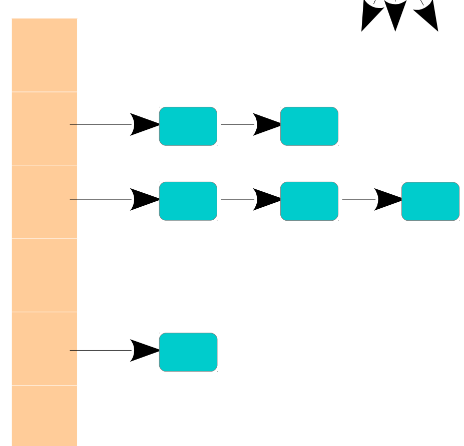
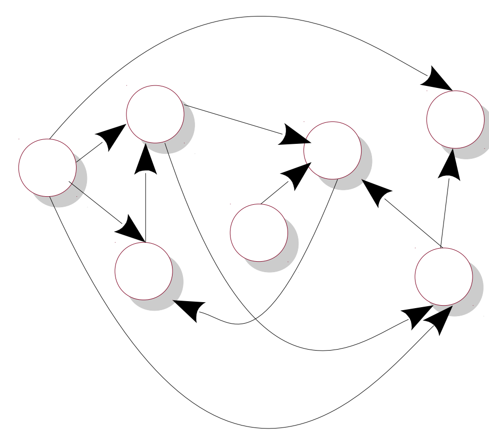
- Generalization of bucket sort.
- Radix sort sorts using different digits.
- At every step, elements are moved to buckets based on their i^{th} digits, starting from the least significant digit.
- **Classwork**: 33, 453, 124, 225, 1023, 432, 2232

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		

Summary

- Array
- Linked List
  - Stack
  - Queue
- Tree
  - Binary Tree
  - Binary Search Tree
  - Heap
  - …
- Hash Table
- Graph

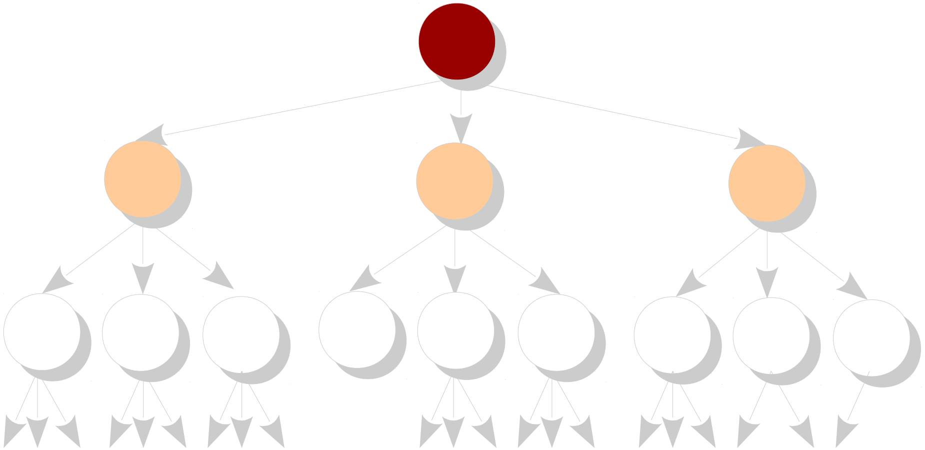


DSAP Usage

- In several applications, arrays (and matrices) suffice. The data is static.
- Most of our data structures are designed for other cases: the data is dynamic.
- Properties of the problem dictate both the algorithm and the associated data structures.
- Algorithms often use data structures as tools.

ID6105: Computational Tools: Algorithms, Data Structures and Programs



B. S. V. Prasad Patnaik, Rupesh Nasre.

August 2021