Arrays

Rupesh Nasre.
rupesh@iitm.ac.in

July 2019

Properties

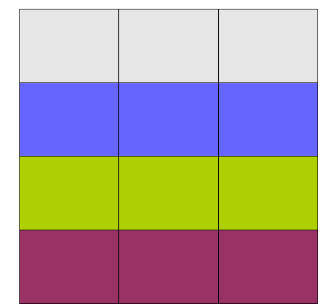
- Simplest data structure
  - Acts as aggregate over primitives or other aggregates
  - May have multiple dimensions
- Contiguous storage
- Random access in O(1)
- Languages such as C use type system to index appropriately
  - e.g., a[i] and a[i + 1] refer to locations based on type
- Storage space: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

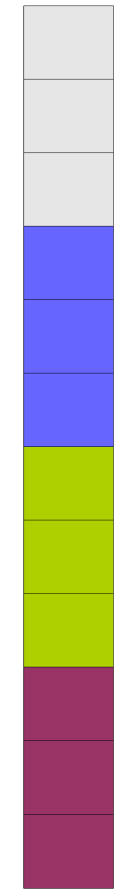
Array Expressions

```
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.



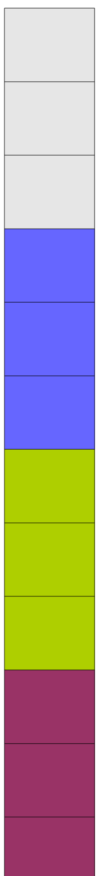


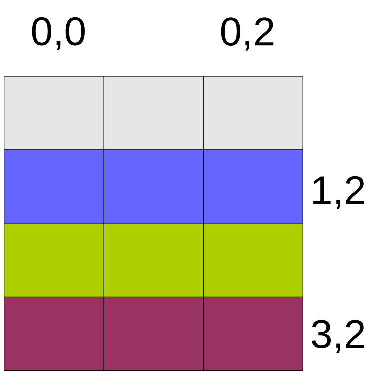
For declaration int a[w4][w3][w2][w1]:

- 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

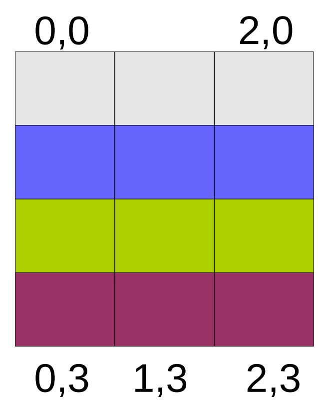
Array Expressions

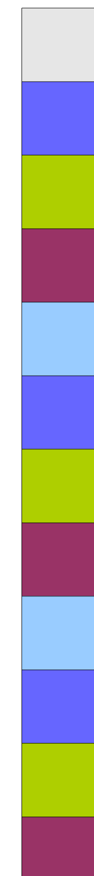
- In C, C++, Java, we use *row-major* storage.
  - All elements of a row are stored together.





- 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.

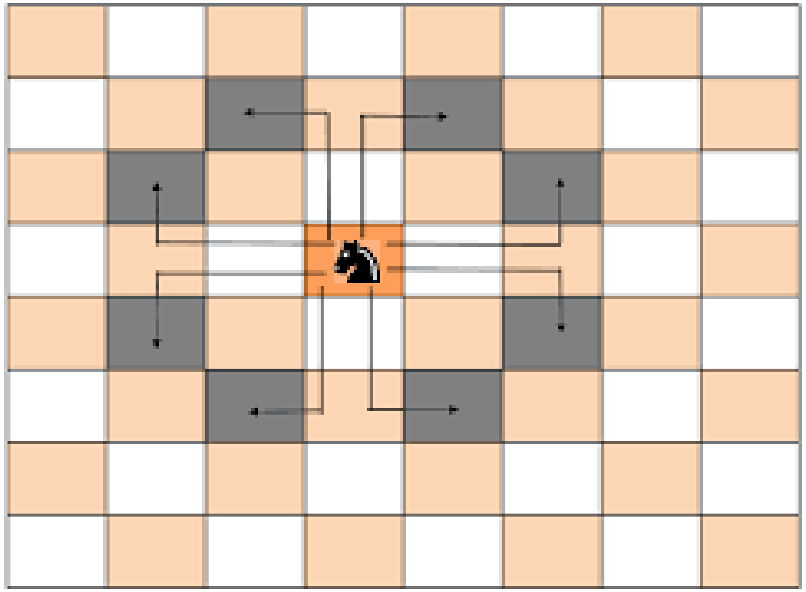
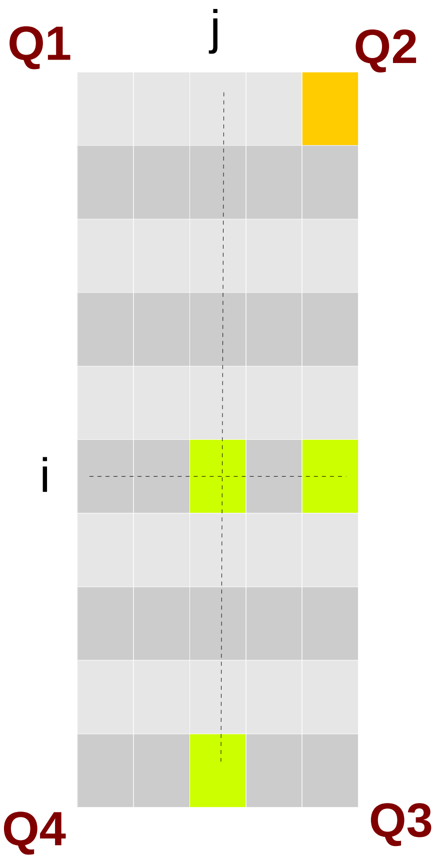


Image source: tutorialhorizon.com

Binary Search in a Sorted Matrix

- 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 → Q2, Q3, Q4
  - $T(M, N) = 3T(M/2, N/2) + c = \min(M, N)^{1.54}$
- Approach 2: Elimination
  - Consider e: 0, N-1.
  - If key < e, eliminate that column
  - If key > e, eliminate that row
  - $O(M + N)$
- Approach 3: Divide and Conquer
  - Use the corner points of Q1, Q2, Q3, Q4 to decide the quadrant.
  - $T(M, N) = 2T(M/2, N/2) + c = O(\min(M, N))$



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

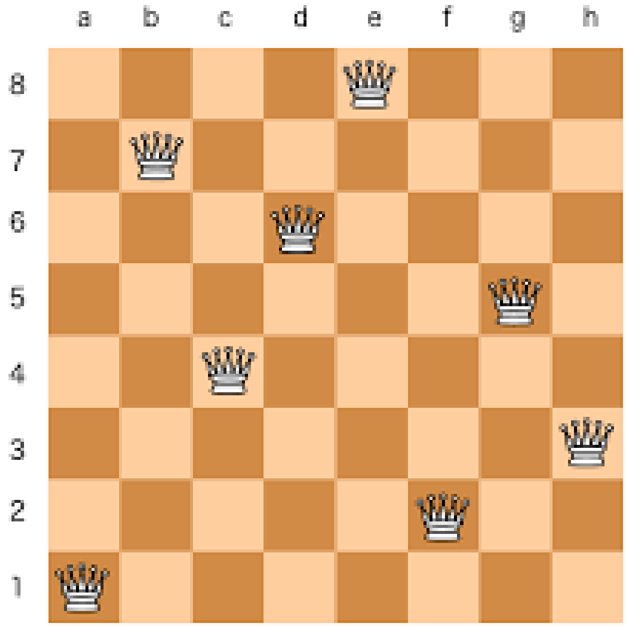
Given a chess-board,

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 α log α)	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?
- **Classwork**: Write the code.

6	2	4	9	11	7	8	1	3	5

Insertion Sort

- Invariant: Keep the first i elements sorted.
- **Classwork**: Write the code.
- Good case, bad case?

6	2	4	9	11	7	8	1	3	5

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.
- **Classwork**: Write the code.

6	2	4	9	11	7	8	1	3	5

Selection Sort

- Approach: Choose the minimum element, and push it to its final place.
- What is the invariant?
- **Classwork:** Write the code.

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.
- Best case, worst case?
- **Classwork:** Write the code.

6	2	4	9	11	7	8	1	3	5

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:** Write the code (reuse the merge function already written).

6	2	4	9	11	7	8	1	3	5

Comparison-based Sorts

- Array consists of n distinct elements.
- Number of permutations = n!
- A sorting algorithm must distinguish between these permutations.
- The number of yes/no bits necessary to distinguish n! permutations is log(n!).
  - Also called information theoretic lower bound
- Given: $N! >= (n/2)^{n/2}$
- log(N!) >= n/2 log(n/2) which is Ω (n log n)
- Comparison-based sort needs 1 bit per comparison (two numbers). Hence it must require at least n log n time.
  - For each comparison-based sorting algorithm, there exists an input for which it would take n log n comparisons.
  - Heapsort, mergesort are theoretically asymptotically optimal (subject to constants)

Bucket Sort

- Hash / index each element into a bucket, based on its value (specific hash function).
- 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.

6	2	4	9	11	7	8	1	3	5

Counting Sort

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

Original array	6	2	4	9	11	7	8	1	3	5
Buckets	1, 2		3	4, 5, 6	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	2	3	4	5	6	7	8	9	11

Radix Sort

- 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		