1. Sorted List Matching Problem
   1. Input: 2 sorted lists of numbers
   2. Output: sorted list of all numbers that appear in both lists
   3. Both lists have unique numbers only
   4. Example
      1. list\_1: 2, 6, 12, 13, 14, 19, 27, 42
         1. Length = *n*
      2. list\_2: 1, 4, 5, 6, 8, 13, 18, 19, 22, 25, 42, 88
         1. Length = *m*
   5. Simple search algorithm
      1. Code
         1. for (i = 0; i < n; i++)
         2. {
            1. for (j = 0; j < m; j++)
            2. {

if (list[i] == list[j])

{

printf(“%d”, list[i]);

break;

}

* + - * 1. }
      1. }
    1. Runtime
       1. 0(*nm*)
  1. Binary search
     1. Code
        1. for (i = 0; i < n; i++)
        2. {
           1. if (binsearch(list\_1, list\_2)
           2. printf(“%d”, list\_1[i]);
        3. }
     2. Runtime
        1. 0(*n* lg *m*)
  2. 3rd algorithm with 2 integer array indexes sweeping left-to-right
     1. Code
     2. Runtime
        1. 0(*n*+*m*)
        2. Most efficient
           1. Less efficient than binary search if *n* or *m* is very large

1. Full code
2. // Arup Guha
3. // Binary Search Example
4. // COP 3502 - Written in class 1/12/2016
5. // Also used to introduce malloc.
6. #include <stdio.h>
7. #include <stdlib.h>
8. #include <time.h>
9. int binarysearch(int searchval, int\* array, int length);
10. int\* makeRandSortedArray(int length);
11. int main() {
12. srand(time(0));
13. /\*\*\* I CAN"T DO THIS **Variable Length Array**
14. int n;
15. scanf("%d", &n);
16. int numbers[n];
17. \*\*\*/
18. // Do this instead:
19. int n, i;
20. printf("How big do you want your array?\n");
21. scanf("%d", &n);
22. // Create a sorted array of numbers.
23. int\* numbers = makeRandSortedArray(n);
24. // Search for first 100 numbers.
25. for (i = 0; i<100; i++)
26. if (binarysearch(i, numbers, n))
27. printf("Found %d\n", i);
28. for (i=0; i<30; i++)
29. printf("%d ", numbers[i]);
30. printf("\n");
31. free(numbers);
32. return 0;
33. }
34. // Fills numbers[0..length-1] with random integers in sorted order.
35. int\* makeRandSortedArray(int length) {
36. int i;
37. // Allocate space.
38. int\* numbers = malloc(sizeof(int)\*length);
39. // Fill the array.
40. numbers[0] = rand()%20;
41. for (i=1; i<length; i++)
42. numbers[i] = numbers[i-1] + rand()%10 + 1;
43. // Return a pointer to the front of the array.
44. return numbers;
45. }
46. // Precondition: array is sorted.
47. // Returns 1 iff array[0..length-1] contains searchval. Returns 0 otherwise.
48. int binarysearch(int searchval, int\* array, int length) {
49. int low = 0, high = length-1;
50. // Search while there is a valid search space.
51. while (low <= high) {
52. int mid = (low+high)/2;
53. // Value is too small.
54. if (searchval < array[mid])
55. high = mid-1;
56. // too big.
57. else if (searchval > array[mid])
58. low = mid+1;
59. // found it!
60. else
61. return 1;
62. }
63. // Never found it.
64. return 0;
65. }
66. Problem: size *nm* matrix of all 1s followed by all 0s
    1. Question: what is the max number of 1s in a row?
    2. Algorithms
       1. Double for loop
          1. Runtime
             1. 0(*nm*)
       2. Check row by row
          1. Inefficient
          2. Runtime
             1. 0(*nm*)
       3. Trace down on the 0s
          1. Find the first 0 on the first row
          2. Go vertically down to find the first 1
          3. Runtime
             1. 0(*n* + *m*)
          4. Code
             1. //Creates an *nm* grid of 1s and 0s with all 1s followed by all 0s
             2. int\*\* makeOnesGrid (int n, int m){

//Allocating n pointers to int

int\*\* grid = malloc(n \* sizeof(int\*));

//For each pointer, allocate an array of *m* integers

int i;

for (i = 0; i < n; i++) {

grid[i] = malloc(m \* sizeof(int));

int numOnes = rand()%(m+1);

//Put 1s

for (j = 0; j < numOnes; j++)

grid[i][j] = 1;

//Put all rest 0s

for (j = numones; j < m; j++)

grid[i] [j] = 0;

}

int maxOnes (int \*\* grid, int n, int m){

1. Freeing
   1. Free things in backwards order of malloc()s.
      1. Including loops