Ch 8. Searching and Sorting Arrays
Part 1

CS 2308
Fall 2011

Jill Seaman

Lecture 1

Definitions of Search and Sort

- Search: find an item in an array, return the index to the item, or -1 if not found.
- Sort: rearrange the items in an array into some order (smallest to biggest, alphabetical order, etc.).
- There are various methods (algorithms) for carrying out these common tasks.
- Goal: compare the efficiency of the algorithms.

Linear Search

- Very simple method.
- Compare first element to target value,
  if not found then compare second element to target value . . .
- Repeat until target value is found
  (return its index) or
  we run out of items (return -1).

Linear Search in C++

```
int searchList (int list[], int numElems, int value) {

  int index=0;           //index to process array
  int position = -1;     //record position of value
  bool found = false;    //flag, true when value is found

  while (index < numElems && !found)
  {
    if (list[index] == value)  //found the value!
    {
      found = true;            //set the flag
      position = index;        //record which item
    }
    index++;                   //increment loop index
  }
  return position;
}
```

Questions

- Why not a for loop?
- Why return -1 for not found?

Program using Linear Search

```
#include <iostream>
using namespace std;

int searchList(int[], int, int);
const int SIZE=5;

int main() {
  int idNums[SIZE] = {871, 750, 988, 100, 822};
  int results, id;

  cout << "Enter the employee ID to search for: ";
  cin >> id;

  results = searchList(idNums, SIZE, id);

  if (results == -1) {
    cout << "That id number is not registered\n";
  } else {
    cout << "That id number is found at location ";
    cout << results+1 << endl;
  }

  return 0;
}
```

Efficiency of Search Algorithms

- We measure efficiency of algorithms in terms of number of main steps required to finish.
- For search algorithms, the main step is comparing array element to the target value.
- Number of steps depends on:
  - size of input array
  - whether or not value is in array
  - where the value is in the array

Efficiency of Linear Search

	N=50,000	In terms of N
Best Case:	1	1
Average Case:	25,000	N/2
Worst Case:	50,000	N

*N is the number of elements in the array

Note: if we search for items not in the array, the average case will increase.

Binary Search

- Works only for SORTED arrays
- Compare target value to middle element in array.
    - if equal, then return index
    - if less than middle elem, search in first half
    - if greater than middle elem, search in last half
- If search list is narrowed down to 1 elem, and it is not equal to target value, return -1
- Divide and conquer style algorithm

Binary Search Algorithm

```
The algorithm described in pseudocode:

location = -1;
first = 0;
last = number of items in list minus 1;

while ((number of items left to search >= 1) and
       (target not found))
   middle = pos of middle item, ½-way between first and last
   if (item at middle position is target)
        target found
        location = middle
   else
        if (target < middle item)
            search lower half of list next:
            last = middle - 1;
        else
            search upper half of list next:
            first = middle + 1;
end while
```

Binary Search in C++

```
int binarySearch (int array[], int numElems, int value) {

  int first = 0,              //index to first elem
      last = numElems – 1,    //index to last elem
      middle,                 //index of middle elem
      position = -1;          //index of target value
  bool found = false;         //flag

  while (!found && first <= last) {

    middle = (first + last) /2;    //calculate midpoint

    if (array[middle] == value) {
      found = true;
      position = middle;
    } else if (array[middle] > value) {
      last = middle – 1;           //search lower half
    } else {
      fisrt = middle + 1;          //search upper half
    }
  }
  return position;
}
```


Program using Binary Search

```
#include <iostream>
using namespace std;

int binarySearch(int[], int, int);
const int SIZE=5;

int main() {
  int idNums[SIZE] = {100, 750, 822, 871, 988};
  int results, id;

  cout << "Enter the employee ID to search for: ";
  cin >> id;

  results = binarySearch(idNums, SIZE, id);

  if (results == -1) {
    cout << "That id number is not registered\n";
  } else {
    cout << "That id number is found at location ";
    cout << results+1 << endl;
  }

  return 0;
}
```

Efficiency of Binary Search

Calculate worst case for N=1024

Num items left to search	Comparisons so far
1024	0
512	1
256	2
128	3
64	4
32	5
16	6
8	7
4	8
2	9
1	10

If $1024 = 2^{10}$ then what does 10 = ?

Efficiency of Binary Search

	N=50,000	In terms of N
Best Case:	1	1
Worst Case:	15.6	$\log_2 N$

*N is the number of elements in the array

Is $\log_2 N$ (binary search) better than N (linear search)?

[Is it really fair to compare these two algorithms?]

Is Log_2 N better than N?

Compare values of N/2, N, and Log_2 N as N increases:

N	N/2	Log_2 N
5	2.5	2.3
50	25	5.6
500	250	9
5,000	2,500	12.3
50,000	25,000	15.6

observation: n/2 is growing much faster than log n!

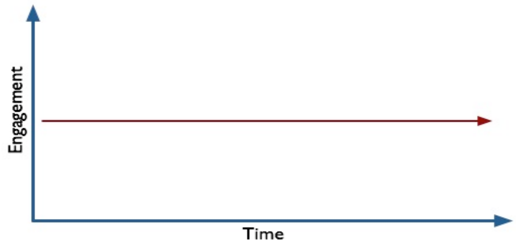
slower growing is more efficient.

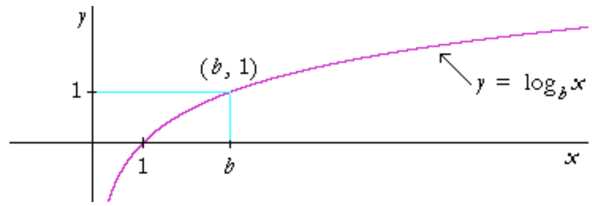
Classifications of (math) functions

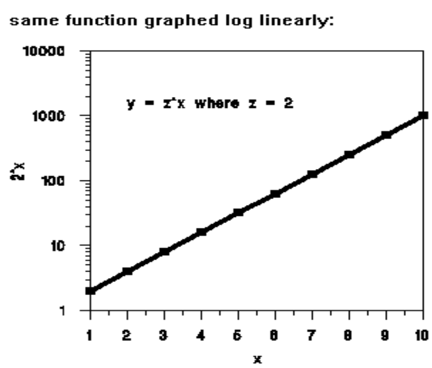
• Constant	y=b	O(1)
• Logarithmic	$y=\log_b(x)$	O(log n)
• Linear	y=ax+b	O(n)
• Linearithmic	$y=x \log_b(x)$	O(n log n)
• Quadratic	$y=ax^2+bx+c$	$O(n^2)$
• Exponential	$y=b^x$	$O(2^x)$

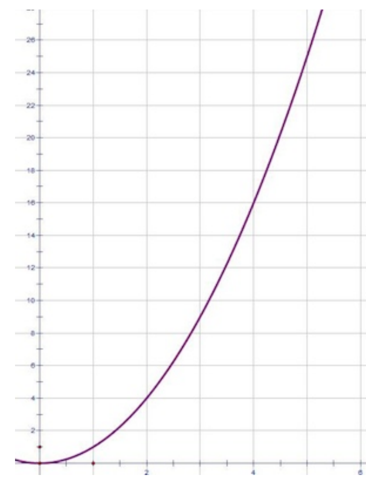
Last column is “big Oh notation” used in CS

Some graphs

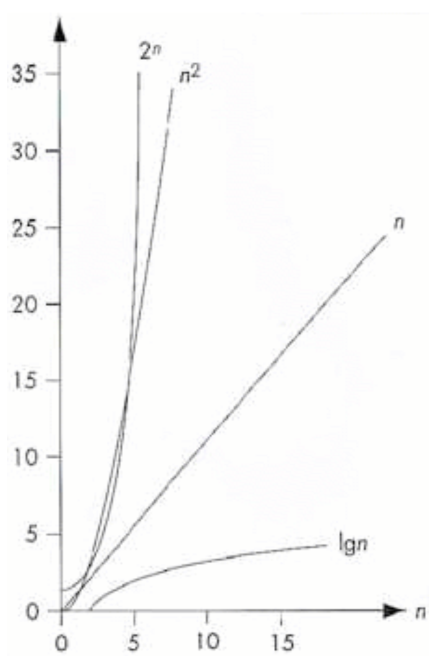


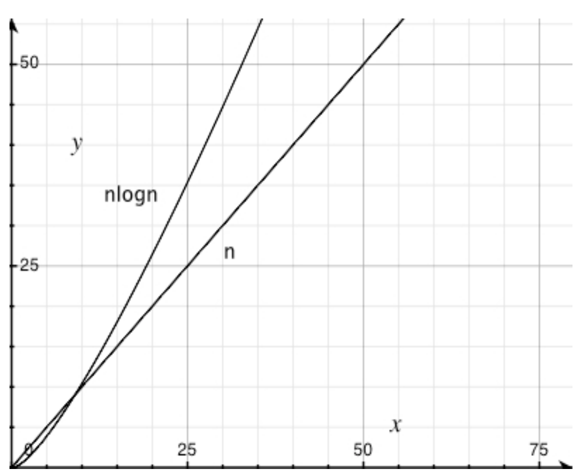






Some more graphs





Efficiency of Algorithms

- To classify efficiency of an algorithm:
  - Express “time” as a function of input
  - Determine which classification the function fits into.
- Nearer to the top is slower growth, and more efficient (constant is better than logarithmic, etc.)