**In class exercise: (assume n 🡪 infinity)**

1. **Linear search**

Ip: List of sorted (or not sorted) elements + search item.

Op: found or not found

Ex. 5, 7, 9, 11, 23, 47

Search\_item = 23 or search\_item = -3

B(n) = \_\_\_\_\_\_\_\_\_\_\_\_\_ (best case when \_\_\_\_\_\_\_\_\_1styugyuguhiui\_\_\_\_\_\_\_\_\_\_)

W(n) = \_\_\_\_\_\_\_\_\_\_\_\_\_ (worst case when \_\_\_\_\_\_\_last + \_\_\_\_\_\_\_\_\_\_)

O(n) = A(n) = \_\_\_\_\_\_\_\_\_\_\_\_\_ (average case when \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

**T/F** If search\_item is in the list, the linear search order is the same for both sorted and unsorted list.

1. **Binary search**

Ip: List of sorted n = 2^k items + search item.

Op: found or not found

Ex. 1, 2, 3, 4, 5, 6, 7, 8

Search\_item = -5

|  |  |
| --- | --- |
| 1, 2, 3, 4, 5, 6, 7, 8 | n=8=2^3=2^k 🡺 k=3 in our case |
| -5 == 4 ?  1, 2, 3, 4 | n/2 = 8/2 = 4 |
| -5 == 2 ?  1, 2 | n/2^2 = 8/4 = 2 |
| -5 == 1 ?  Op: not found (I did 3 comparisons)  🡺 For a list of size n=2^k 🡺 k divisions and comparisons 🡺 k=log2(n) 🡺 O(log n) | n/2^3 = 8/8 = 1 |

B(n) = \_\_\_\_\_\_\_\_\_\_\_\_\_ (best case when \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

W(n) = \_\_\_\_\_\_\_\_\_\_\_\_\_ (worst case when \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

A(n) = \_\_\_\_\_\_\_\_\_\_\_\_\_ (average case when \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

1. How many comparisons of array items do the following loops contain?

for (j = 1; j <= n - 1; j++)

{

i = j + 1;

do

{

if (theArray[i] < theArray[j])

swap(theArray[i], theArray[j]);

i++;

} while (i <= n);

}

1. What order is an algorithm that has as a growth-rate function

**a.** 8 x n3 – 9 x n O(…. )

**b.** 7 x log2 n+ 20

**c.** 7 x log2 n+ *n*

1. [Optional, if you have time] Create a CLion project and copy and paste above main() the recBinarySearch method. Fix it and modify it to its template version. Then use it to search a number in an array of ints and to search a word in an array of strings. A possible main function:

bool fnd;

int location;

int a[] = {11,22,33,44,55,66,77,88,99};

recBinarySearch(a, 0, 8, 66, fnd, location);

cout << "Item 66 found? " << boolalpha << fnd << " at location: "<< location <<endl;

string names[] = {"ana","bob", "joe", "kitty", "tom"};

string s = "bob";

recBinarySearch(names, 0, 4, s, fnd, location);

cout << "Item 'bob' found? " << boolalpha << fnd << " at location: "<< location <<endl;

void recBinarySearch (ArrayType a, int first, int last,

ElementType item,

bool & found, int & loc)

/\*-------------------------------------------------------

Recursively search sub(list) a[first] , ... a[last]

for item using a binary search.

Precondition: Elements of a are in ascending order;

item has the same type as the array elements

Postcondition: found = true and loc = position of item

if search is successful; otherwise, found is false.

-----------------------------------------------------------\*/

{

if (first > last) // anchor 1 -- empty sublist

found = false;

else // inductive case

{ // recursively search

loc = (first + last) / 2;

if (item < a[loc]) // the first half

recBinarySearch( a, first, loc-1, found loc);

else if (item > a[loc]) // the seonc half

recBinarySearch (a, loc + 1, last, found, loc);

else

found = true; // anchor 2 -- found item

}

}