**School of Computer Science**

**CIS\*2520: Data Structures**

**Fall 2024, Lab 5**

**Weeks of 8 and 9, Oct. 28 to Nov. 8**

# Search Algorithms

1. Binary search requires the array to be \_\_\_\_\_\_\_\_\_\_\_\_.

Answer: sorted

Explanation: Binary search assumes that elements are sorted. It is required for the algorithm to be able to divide the search space.

1. Binary search algorithm uses which of the following approaches?
   1. Linear
   2. Sort and search
   3. Greedy approach
   4. Divide and conquer

Explanation: Binary search divides the search space to find the element. Such approach is called divide and conquer.

1. What is the main difference between Interpolation Search and Binary Search?

Explanation: Interpolation search does not start with middle element. Instead, it uses the uniform distribution to interpolate the location of the search element and follows the divide and conquer approach if not found in the first shot.

1. Consider the recursive implementation of Binary Search algorithm from the lecture notes. Given the array a=[2,4,7,12,23,34,42,45,67,133,145,156,178] and two values x=42 and y=177. Calculate how many times we need to call the recursive function BinarySearch find x and y in the array a?

Answer: x is a mid-point of the array, hence value will be found in the first call. Y is not in the array, hence we need log2(n) calls to verify that. log2(13)~=3.7 hence we need 4 calls to find out that y is not in the array.

1. Consider the following function. It attempts to find element x in a sorted array a of n elements (n>1). This function contains an error. Identify the cases when the function fails and attempt to fix it.

1 void fun(int n, int a[], int x){

2 int i,j,k;

3

4 i= 0;

5 j= n-1;

6 do{

7 k=(i+j) / 2;

8 if (a[k] < x)

9 i = k;

10 else

11 j = k;

12 }

13 while((a[k] != x) && (i < j));

14

14 if (a[k] == x)

16 printf ("x is in the array");

17 else

18 printf ("x is not in the array");

19 }

Explanation: The function is iterative implementation of Binary Search.  k keeps track of current middle element. i and j keep track of left and right corners of current subarray. Calculation of i and j (lines 8-11) is incorrect. Because of that for x >= max(a) the function goes into infinite loop. Correct code for calculatioin of i and j is:

if (x <= a[k])

j = k-1;

if (a[k] <= x)

i = k+1;

1. Implement function ceilSearch that is given a sorted array of integers and a searched element x. The function should return a ceiling of the element x.

The ceiling of an element x is the smallest element present in array which is greater than or equal to x. Ceiling is not present if x is greater than the maximum element present in array. For example, if the given array is {12, 67, 90, 100, 300, 399} and x = 95, then the output should be 100.

What can be the minimum worst-case time complexity to find ceiling of a number x in given array?

* 1. O( log(log(n)) )
  2. O( n )
  3. O( log(n) )
  4. O( log(n)\*log(n) )

Explanation:

We modify the standard binary search to find the ceiling. The solution is O(log(n)).

int ceilSearch(int arr[], int low, int high, int x)

{

int mid;

/\* If x is smaller than or equal to the first element,

then return the first element \*/

if (x <= arr[low])

return low;

/\* If x is greater than the last element, then return -1

\*/

if (x > arr[high])

return -1;

/\* get the index of middle element of arr[low..high]\*/

mid = (low + high) / 2; /\* low + (high - low)/2 \*/

/\* If x is same as middle element, then return mid \*/

if (arr[mid] == x)

return mid;

/\* If x is greater than arr[mid], then either arr[mid +

1] is ceiling of x or ceiling lies in

arr[mid+1...high] \*/

else if (arr[mid] < x) {

if (mid + 1 <= high && x <= arr[mid + 1])

return mid + 1;

else

return ceilSearch(arr, mid + 1, high, x);

}

/\* If x is smaller than arr[mid], then either arr[mid]

is ceiling of x or ceiling lies in arr[mid-1...high]

\*/

else {

if (mid - 1 >= low && x > arr[mid - 1])

return mid;

else

return ceilSearch(arr, low, mid - 1, x);

}

}

Hence Option(C) is the correct answer.

# Hashing

1. Consider the following hash table and a function h(x)=x%10. What will be the bucket of the x=11?

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|  | 21 | 12 | 33 | 14 |  |  |  | 78 |  |

Answer: 5

Explanation: 11%10 = 1, 1 is taken, 2 is taken, 3 is taken, 4, is taken, hence first empty is 5

1. How many different insertion sequences of the key values using the hash function **h(k) = k mod 10** and linear probing will result in the hash table shown below?

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|  |  | 42 | 23 | 34 | 52 | 46 | 33 |  |  |

1. 10
2. 20
3. 30
4. 40

Answer: In a valid insertion sequence, the elements 42, 23 and 34 must appear before 52 and 33, and 46 must appear before 33.

Total number of different sequences = 3! x 5 = 30 -> 46 은 어디에나 존재가능 하지만 33 전에 나와야하기떄문에 5, 3! -> 42 23 34 가 나올수있는 가능성.

In the above expression, 3! is for elements 42, 23 and 34 as they can appear in any order, and 5 is for element 46 as it can appear at 5 different places.

1. Which of the following hash functions is most likely to cause clustering in a hash table?
   1. h(k) = k % m
   2. h(k) = floor(m \* (kA % 1))
   3. h(k) = k
   4. h(k) = ((k / m) + k \* m) + k % m

Answer: The modulo operation in option A results in clustering since the hash values of keys that are close to each other will also be close, leading to collisions. Options B, C, and D use more complex hash functions, which spread out the keys more uniformly across the table. Note: in B, the values of A can be irrational number like pi-3 or (sqrt(5)-1)/2.

1. Which one of the following hash functions on integers will distribute keys most uniformly over 10 buckets numbered 0 to 9 for i ranging from 0 to 2020?  
   1. h(i) = (12 ∗ i) mod 10
   2. h(i) = (11 ∗ i2) mod 10
   3. h(i) =i3 mod 10
   4. h(i) =i2 mod 10

Answer:

Since mod 10 is used, the last digit matters. If you do cube all numbers from 0 to 9, you get following

Number Cube Last Digit in Cube

0 0 0

1 1 1

2 8 8

3 27 7

4 64 4

5 125 5

6 216 6

7 343 3

8 512 2

9 729 9

Therefore all numbers from 0 to 2020 are equally divided in 10 buckets. If we make a table for square, we don\'t get equal distribution. In the following table. 1, 4, 6 and 9 are repeated, so these buckets would have more entries and buckets 2, 3, 7 and 8 would be empty.

Number Square Last Digit in Square

0 0 0

1 1 **1**

2 4 **4**

3 9 **9**

4 16 **6**

5 25 5

6 36 **6**

7 49 **9**

8 64 **4**

9 81 **1**

**Alternative approach -**   
Using the concept of power of cycle:   
  
(a) (0,1,4,9,6,5,6,9,4,1,0) repeated   
(b) (0,1,8,7,4,5,6,3,2,9) repeated   
(c) (0,1,4,9,6,5,6,9,4,1,0) repeated   
(d) (0,2,4,6,8) repeated   
  
So, only h(i) =i3 mod 10 covers all the digits from 0 to 9.   
Hence Option (C) is correct. answer

1. Why it is essential for the search algorithm in the hash table to distinguish between empty-since-start and empty-after-deletion buckets?

Answer: Item may have been placed in a subsequent bucket before this bucket's item was removed. Algorithm stops when empty-since-start or if it finds the element sought for.

1. What will be the state of the hash array with quadratic probing after inserting the following sequence using hash function h(k)=k % 15 (if collision is detected h(k)=k%15+ci2)

Sequence of items to insert: 32, 30, 49, 3, 99, 23, 11, 42, 15, 62

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 30 | 15 | 32 | 3 | 49 |  | 62 |  | 23 | 99 |  | 11 | 42 |  |  |

1. Insert the characters of the string **K R P C S N Y T J M** into a hash table of size 10. Use the hash function

h(x) = ( ord(x) – ord("A") + 1 ) mod10

If linear probing is used to resolve collisions, then the following insertion causes collision

* 1. Y
  2. C
  3. M
  4. P

Answer:

The hash table with size 10 will have index from 0 to 9. hash function = h(x) = ((ord(x) - ord(A) + 1)) mod 10 So for string K R P C S N Y T J M: K will be inserted at index : (11-1+1) mod 10 = 1 R at index: (18-1+1) mod 10 = 8 P at index: (16-1+1) mod 10 = 6 C at index: (3-1+1) mod 10 = 3 S at index: (19-1+1) mod 10 = 9 N at index: (14-1+1) mod 10 = 4 Y at index (25-1+1) mod 10 = 5 T at index (20-1+1) mod 10 = 0 J at index (10-1+1) mod 10 = 0 // first collision occurs. M at index (13-1+1) mod 10 = 3 //second collision occurs. Only J and M are causing the collision.

Final Hash table will be:

0 T

1 K

2 J

3 C

4 N

5 Y

6 P

7 M

8 R

9 S

1. Propose a structure that could be used to represent state of a bucket in the hash table with linear probing. Implement a sample using a table of size 10 and a sequence of 5 elements with at least 1 conflict. h(k)=k%10.

Answer:

#include <stdio.h>

typedef enum State{ESS, EDR, BSY} Mode;

//node has to be able to represent two empty states

struct HashNode{

int data;

Mode state;

};

int h(int k, int n){

return k%n;

}

void insert(int k, struct HashNode A[], int n){

int i = h(k,n);

int p=0;

do{

if(A[i].state==ESS || A[i].state==EDR){

A[i].state=BSY;

A[i].data=k;

return;

}else{

i=(i+1)%n;

p++;

}

}while(p<n);

}

int main() {

struct HashNode table[10];

for(int i=0; i<10; i++){

table[i].data=0;

table[i].state=ESS;

}

int seq[] = {13,42,32,41,15};

for(int i=0; i<5; i++){

insert(seq[i], table, 10);

}

for(int i=0; i<10; i++){

printf("%d, ", table[i].data);

}

return 0;

}