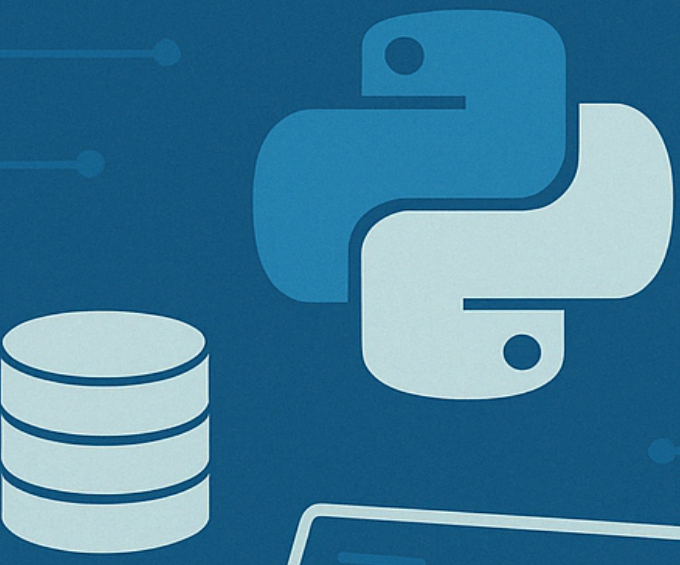
KARNATAKA SECOND PUC

COMPUTER SCIENCE

STUDY MATERIAL

MOHAMMED MATHEEN L R



PYTHON & MYSQL

Contents

LICENSE 3

CHAPTER 6: SEARCHING 4

CHAPTER NOTES . . . 4

6.1 Introduction . . . 4

6.2 Linear Search (Sequential Search) . . . 4

Algorithm 6.1: Linear Search . . . 4

Example . . . 4

6.3 Binary Search . . . 5

Algorithm 6.2: Binary Search . . . 5

Example . . . 6

Program 6-2: Binary Search in Python . . . 6

6.3.1 Applications of Binary Search . . . 7

6.4 Search by Hashing . . . 7

Example . . . 7

Program 6-3: Hashing in Python . . . 8

6.4.1 Collision . . . 9

Summary . . . 9

MULTIPLE CHOICE QUESTIONS (MCQs) 9

FILL IN THE BLANKS 15

2 MAKRS QUESTIONS 16

3 MARKS QUESTIONS 18

5 MARKS QUESTIONS 21

CHAPTER END EXERCISES 25

LICENSE

Figure 1: Licence

CHAPTER 6: SEARCHING

CHAPTER NOTES

6.1 Introduction

- Searching is the process of locating a specific element (called the **key**) in a collection.
- It determines **whether the key is present** and, if so, **its position**.
- Three main techniques:

1. **Linear Search**
2. **Binary Search**
3. **Search by Hashing**

6.2 Linear Search (Sequential Search)

- It compares each element of the list with the key until a match is found or the list ends.
- Useful for **unsorted** or **small-sized** lists.

Algorithm 6.1: Linear Search

```
LinearSearch(numList, key, n)
Step 1: SET index = 0
Step 2: WHILE index < n, REPEAT Step 3
Step 3: IF numList[index] = key THEN
           PRINT "Element found at position", index+1
           STOP
        ELSE
           index = index + 1
Step 4: PRINT "Search unsuccessful"
```

Example

List: `numList = [8, -4, 7, 17, 0, 2, 19]`, Key: `17`

Index	Comparison	Result
0	8 == 17	not found

Index	Comparison	Result
1	-4 == 17	not found
2	7 == 17	not found
3	17 == 17	Found at index 3

```
def linearSearch(list, key):
    for index in range(len(list)):
        if list[index] == key:
            return index + 1
    return None

list1 = []
maximum = int(input("How many elements in your list? "))
for i in range(maximum):
    n = int(input())
    list1.append(n)

key = int(input("Enter the number to be searched:"))
position = linearSearch(list1, key)

if position is None:
    print("Number", key, "is not present in the list")
else:
    print("Number", key, "is present at position", position)
```

6.3 Binary Search

- Requires a **sorted** list.
- More efficient than linear search.
- Repeatedly divides list into halves and compares key with the **middle element**.

Algorithm 6.2: Binary Search

```
BinarySearch(numList, key)
Step 1: SET first = 0, last = n - 1
Step 2: WHILE first <= last, REPEAT Step 3
Step 3: SET mid = (first + last)//2
        IF numList[mid] == key THEN
```

```
            PRINT "Element found at position", mid + 1
            STOP
         ELSE IF numList[mid] > key THEN
            last = mid - 1
         ELSE
            first = mid + 1
Step 4: PRINT "Search unsuccessful"
```

Example

List: `numList = [2,3,5,7,10,11,12,17,19,23,29,31,37,41,43]`, Key: 2 Here's the corrected and neatly formatted table in Markdown:

Step	Low	High	Mid	Mid Value	Comparison
1	0	14	7	17	$2 < 17 \rightarrow$ left half
2	0	6	3	7	$2 < 7 \rightarrow$ left half
3	0	2	1	3	$2 < 3 \rightarrow$ left half
4	0	0	0	2	Match found

Minimum iterations: 1 (if key is in middle)
Maximum iterations: $\lfloor log_2(n) \rfloor$

Program 6-2: Binary Search in Python

```python
def binarySearch(list, key):
    first = 0
    last = len(list) - 1
    while first <= last:
        mid = (first + last)//2
        if list[mid] == key:
            return mid
        elif key > list[mid]:
            first = mid + 1
        else:
            last = mid - 1
    return -1
```

```
numList = []
print("Enter elements in ascending order (-999 to stop):")
num = int(input())
while num != -999:
    numList.append(num)
    num = int(input())

key = int(input("Enter the number to be searched: "))
pos = binarySearch(numList, key)
if pos != -1:
    print(key, "is found at position", pos + 1)
else:
    print(key, "is not found in the list")
```

6.3.1 Applications of Binary Search

- Dictionary/telephone directory search.
- Database indexing.
- Data compression, routing tables, etc.

6.4 Search by Hashing

- Direct access method using a **hash function**.
- Searches key in constant time O(1).
- Hash function computes index:
  `h(element) = element % size_of_table`

Example

List: `[34, 16, 2, 93, 80, 77, 51]`, Hash Table size = 10

Element	Hash Value	Index in Table
34	4	4
16	6	6
2	2	2
93	3	3
80	0	0

Element	Hash Value	Index in Table
77	7	7
51	1	1

Final Hash Table:

Index	Value
0	80
1	51
2	2
3	93
4	34
5	None
6	16
7	77
8	None
9	None

Program 6-3: Hashing in Python

```
def hashFind(key, hashTable):
    if hashTable[key % 10] == key:
        return (key % 10) + 1
    else:
        return None

hashTable = [None]*10
L = [34, 16, 2, 93, 80, 77, 51]

for i in L:
    hashTable[i % 10] = i

key = int(input("Enter the number to be searched: "))
```

```
position = hashFind(key, hashTable)
if position:
    print("Number", key, "present at", position, "position")
else:
    print("Number", key, "is not present in the hash table")
```

6.4.1 Collision

- Occurs when multiple elements hash to the same index.
- Requires **collision resolution** techniques (not covered here).
- A **perfect hash function** prevents collision by mapping each element uniquely.

Summary

Technique	Best For	List Requirement	Time Complexity	Notes
Linear Search	Small/Unordered lists	None	O(n)	Simple but slow for large n
Binary Search	Large/Sorted lists	Sorted	O(log n)	Fast, requires sorted input
Hashing	Fastest lookups	Hash Function	O(1)	Needs collision handling

MULTIPLE CHOICE QUESTIONS (MCQs)

1. What is the primary goal of searching in computer science?
A) To delete elements from a list
B) To sort a list
C) To locate a specific element in a list
D) To multiply elements in a list
Answer: C

2. Which of the following is another name for linear search?
A) Quick search
B) Divide and conquer search
C) Serial search
D) Binary search
Answer: C

3. In linear search, how many comparisons are required in the worst case for a list of n elements?

A) n/2

B) n

C) $log_2 n$

D) 1

Answer: B

4. Linear search is most suitable when the list is:

A) Sorted

B) Encrypted

C) Small and unsorted

D) Large and ordered

Answer: C

5. Which loop is used in the linear search algorithm provided in the chapter?

A) for loop

B) while loop

C) do-while loop

D) nested loop

Answer: B

6. In which case will linear search make only one comparison?

A) When the key is the middle element

B) When the list is sorted

C) When the key is the last element

D) When the key is the first element

Answer: D

7. In linear search, if the key is not present in the list, the number of comparisons will be:

A) 0

B) 1

C) n

D) n-1

Answer: C

8. What is returned by the `linearSearch()` function if the key is not found?

A) 0

B) -1

C) None

D) The index of last element

Answer: C

9. Binary search works only on:

A) Encrypted lists

B) Hashed lists

C) Sorted lists

D) Lists with no duplicates

Answer: C

10. What is the mid index calculated as in the binary search algorithm?

A) (first + last) / 2

B) (first + last) % 2

C) (first + last) // 2

D) (first + last) * 2

Answer: C

11. In the best case, binary search takes how many iterations to find the key?

A) 0

B) 1

C) $log_2 n$

D) n

Answer: B

12. What happens when the middle element in binary search equals the key?

A) The list is reversed

B) The key is ignored

C) The search continues

D) The key is found and search terminates

Answer: D

13. What is the time complexity of binary search in the worst case?

A) $O(n^2)$

B) O(n)

C) O(log n)

D) O(1)

Answer: C

14. Binary search reduces the list size by:

A) One element each time

B) Two elements each time

C) Half each time

D) None of the above

Answer: C

15. What is a requirement for the binary search algorithm to work properly?

A) The list must be hashed

B) The list must be unsorted

C) The key must be negative

D) The list must be sorted

Answer: D

16. What does hashing use to calculate the position of an element?

A) Key length

B) Sorting function

C) Hash function

D) Index counter

Answer: C

17. What is the formula for the remainder method used in hashing?

A) element // table size

B) element % table size

C) element + table size

D) element - table size

Answer: B

18. What is the term used when two elements in hashing map to the same index?

A) Mapping

B) Modulo clash

C) Collision

D) Duplication

Answer: C

19. What is a perfect hash function?

A) A function that sorts a list

B) A function that encrypts elements

C) A function that generates only even indices

D) A function that maps all elements to unique indices

Answer: D

20. Which search technique is fastest in theory when there are no collisions?

A) Linear search

B) Binary search

C) Hash-based search

D) Sequential search

Answer: C

21.

Assertion (A): Linear search is suitable for small and unsorted lists.

Reason (R): Linear search checks every element sequentially from the beginning of the list.

Options:

A) Both A and R are true, and R is the correct explanation of A.

B) Both A and R are true, but R is not the correct explanation of A.

C) A is true, but R is false.

D) A is false, but R is true.

Answer: A

22.

Assertion (A): Linear search performs the minimum work when the key is at the end of the list.

Reason (R): Linear search stops after finding the first occurrence of the key.

Options:

A) Both A and R are true, and R is the correct explanation of A.

B) Both A and R are true, but R is not the correct explanation of A.

C) A is false, but R is true.

D) Both A and R are false.

Answer: C

23.

Assertion (A): Binary search is faster than linear search on large datasets.

Reason (R): Binary search reduces the list size by half in every iteration.

Options:

A) Both A and R are true, and R is the correct explanation of A.

B) Both A and R are true, but R is not the correct explanation of A.

C) A is false, but R is true.

D) Both A and R are false.

Answer: A

24.

Assertion (A): Binary search works efficiently on unsorted data.

Reason (R): Binary search compares the key only with the first and last elements.

Options:

A) Both A and R are true, and R is the correct explanation of A.

B) A is true, but R is false.

C) A is false, but R is true.

D) Both A and R are false.

Answer: D

25.

Assertion (A): In binary search, if the key is not found, the list size continues to reduce until one element remains.

Reason (R): Binary search stops only when `first > last`.

Options:

A) Both A and R are true, and R is the correct explanation of A.

B) Both A and R are true, but R is not the correct explanation of A.

C) A is true, but R is false.

D) Both A and R are false.

Answer: A

26.

Assertion (A): Hashing provides constant time search irrespective of the list size.

Reason (R): Hashing directly accesses the element by calculating its position using a hash function.

Options:

A) Both A and R are true, and R is the correct explanation of A.

B) Both A and R are true, but R is not the correct explanation of A.

C) A is false, but R is true.

D) Both A and R are false.

Answer: A

27.

Assertion (A): A perfect hash function guarantees that no two elements map to the same index.

Reason (R): Perfect hash functions are collision-prone.

Options:

A) Both A and R are true, and R is the correct explanation of A.

B) Both A and R are true, but R is not the correct explanation of A.

C) A is true, but R is false.

D) Both A and R are false.

Answer: C

28.

Assertion (A): Binary search modifies the list by deleting elements that are not required.

Reason (R): The search area reduces by half in each iteration.

Options:

A) Both A and R are true, and R is the correct explanation of A.

B) A is true, but R is false.

C) A is false, but R is true.

D) Both A and R are false.

Answer: C

29.
Assertion (A): Collisions in hashing occur when two keys are stored at the same index.
Reason (R): The modulo division method always avoids collisions.
Options:
A) Both A and R are true, and R is the correct explanation of A.
B) Both A and R are true, but R is not the correct explanation of A.
C) A is true, but R is false.
D) Both A and R are false.
Answer: C

30.
Assertion (A): Binary search is efficient only if the list is sorted.
Reason (R): Sorting allows binary search to determine which half to discard.
Options:
A) Both A and R are true, and R is the correct explanation of A.
B) Both A and R are true, but R is not the correct explanation of A.
C) A is false, but R is true.
D) Both A and R are false.
Answer: A

FILL IN THE BLANKS

1. The process of locating a particular element in a collection of elements is called _____.
Answer: Searching

2. In linear search, each element is compared with the key in a _____ manner.
Answer: sequential

3. Another name for linear search is _____ search.
Answer: serial

4. The maximum number of comparisons in linear search is equal to _____, where n is the number of elements.
Answer: n

5. In binary search, the list must be _____ before applying the search.
Answer: sorted

6. Binary search divides the list into _____ parts after every unsuccessful comparison.
Answer: two / halves

7. The mid index in binary search is calculated as _____.
Answer: (first + last) // 2

8. Binary search reduces the search area by _____ each time.
Answer: half

9. In hashing, the formula used to compute index using the remainder method is _____.
Answer: element % size of hash table

10. The table where elements are stored using hash function values is called a _____.
Answer: hash table

11. A situation where two or more elements map to the same index in a hash table is called a _____.
Answer: collision

12. The process of finding a new position for elements during collision is known as _____.
Answer: collision resolution

13. A hash function that maps every key to a unique index is called a _____ hash function.
Answer: perfect

14. The time complexity of binary search in the worst case is _____.
Answer: O(log n)

15. In the worst case of linear search, the key may be found at the _____ of the list or not present at all.
Answer: end

2 MAKRS QUESTIONS

1. What is linear search? When is it preferred?
Answer:
Linear search is an exhaustive technique where each element in the list is compared sequentially with the key until the element is found or the list ends.
It is preferred for **small, unsorted lists**.

2. What is binary search? State its requirement.
Answer:
Binary search is an efficient search method that repeatedly divides a **sorted** list into halves and compares the key with the middle element to find its position.
It requires the list to be **sorted in ascending or descending order**.

3. Differentiate between Linear Search and Binary Search.
Answer:

Linear Search	Binary Search
Works on unsorted or sorted lists	Works only on sorted lists
Time complexity is O(n)	Time complexity is O(log n)
Compares each element sequentially	Divides list and compares with middle

4. Give the syntax of a simple hash function and one example using element = 34, table size = 10.
Answer:
Syntax: `hash_value = element % table_size`
Example: `34 % 10 = 4` → So, element 34 will be stored at index 4 of the hash table.

5. What is a hash table? How is it implemented in Python?
Answer:
A hash table is a list-like structure where elements are stored at specific positions determined by a hash function.
In Python, it can be implemented using a **list** with predefined size, e.g.:

```
hashTable = [None] * 10
```

6. Define 'collision' in hashing. What is a perfect hash function?
Answer:
Collision occurs when two or more elements produce the same hash value and thus map to the same index.
A perfect hash function maps every input to a **unique index**, ensuring **no collisions**.

7. How does binary search minimize the number of comparisons during a search?
Answer:
Binary search reduces the list size by half after each comparison. By checking whether the key is less than or greater than the middle element, it eliminates one half of the list in each iteration.

8. Write the output of the following:

```
L = [12, 23, 3, -45]
key = 23
position = linearSearch(L, key)
print(position)
```

Answer:
The key 23 is found at position **2** (index + 1).
So the output is:
2

9. List any two applications of binary search.

Answer:

1. Searching a word in a dictionary or telephone directory
2. Implementing indexing in databases and routing tables in routers

10. Give any two differences between binary search and hashing.

Answer:

Binary Search	Hashing
Works only on sorted lists	Works on any list (hash function based)
Requires multiple comparisons (log n)	Ideally needs only one comparison

3 MARKS QUESTIONS

1. Explain the working of linear search with an example.

Answer:

Linear search compares each element in the list with the key from beginning to end.

For example, in the list `numList = [8, -4, 7, 17, 0, 2, 19]`, to find key `17`, the comparisons will be:

Index	Value	Comparison
0	8	8 ≠ 17
1	-4	-4 ≠ 17
2	7	7 ≠ 17
3	17	17 = 17 □

Search is successful at **position 4**.

2. Write the algorithm for Binary Search and explain its steps.

Answer:

Algorithm:

```
BinarySearch(numList, key)
Step 1: SET first = 0, last = n - 1
```

```
Step 2: WHILE first <= last
    mid = (first + last)//2
    IF numList[mid] == key
        PRINT "Element found at position", mid+1
        STOP
    ELSE IF numList[mid] > key
        last = mid - 1
    ELSE
        first = mid + 1
Step 3: PRINT "Search unsuccessful"
```

Explanation:
The list is divided repeatedly. Based on comparison with the middle value, only one half is searched in each iteration.

3. Differentiate between Linear Search, Binary Search, and Hashing (any three points).
Answer:

Feature	Linear Search	Binary Search	Hashing
Input Requirement	Unsorted/Sorted	Must be sorted	No order required
Comparisons	O(n)	O(log n)	O(1) ideally
Applications	General search	Efficient numeric search	Fast lookup, e.g., dictionaries
proach	Sequential	Divide-and-Conquer	Direct computation

4. What is a hash function? How is it used to create a hash table? Give an example.
Answer:
A hash function computes an index using a mathematical formula.
Example: `h(element) = element % size_of_table`
For `element = 93` and `table size = 10`,
`hash value = 93 % 10 = 3`
This means the element will be placed at index 3 of the hash table.

5. Describe how binary search performs when the key is the first element in the list. Use a dry run to explain.
Answer:
For list `numList = [2,3,5,7,10,11,12,17,19,23,29,31,37,41,43]`, key = 2:

- Iteration 1: mid = 7 → 17 > 2 → search left
- Iteration 2: mid = 3 → 7 > 2 → search left
- Iteration 3: mid = 1 → 3 > 2 → search left
- Iteration 4: mid = 0 → 2 == 2 → found

Search completes in **4 iterations**.

6. What is collision in hashing? Why does it occur? Explain with an example.

Answer:

Collision occurs when two different elements are assigned the same hash index.

Example:

Using `h(element) = element % 10`,

`16 % 10 = 6` and `26 % 10 = 6` → both map to index 6

This causes a collision since both elements can't occupy the same slot.

7. Give the Python code for linear search and explain its working.

Answer:

```
def linearSearch(list, key):
    for index in range(len(list)):
        if list[index] == key:
            return index + 1
    return None
```

Explanation:

- The function iterates through the list.
- If `key == list[index]`, it returns the position (index+1).
- If key is not found, it returns `None`.

8. List three advantages of binary search over linear search.

Answer:

1. **Faster for large lists:** It reduces comparisons using a divide-and-conquer method.
2. **Time complexity is O(log n):** More efficient than linear search's O(n).
3. **Fewer comparisons:** Especially when the key is near the center of a sorted list.

5 MARKS QUESTIONS

1. Explain the working of linear search algorithm with the help of Algorithm and a dry run example.
Answer:
Linear search sequentially compares each element in the list with the key until it is found or the list ends.

Algorithm 6.1: Linear Search

```
LinearSearch(numList, key, n)
Step 1: SET index = 0
Step 2: WHILE index < n, REPEAT Step 3
Step 3: IF numList[index] = key THEN
           PRINT "Element found at position", index+1
           STOP
        ELSE
           index = index + 1
Step 4: PRINT "Search unsuccessful"
```

Dry Run Example:
List: `numList = [8, -4, 7, 17, 0, 2, 19]`, key = 17
→ 4 comparisons are made to find key 17 at index 3 (position 4).

2. Describe the binary search algorithm with its working and dry run using Algorithm
Answer:

Algorithm 6.2: Binary Search

```
BinarySearch(numList, key)
Step 1: SET first = 0, last = n-1
Step 2: mid = (first + last)//2
Step 3: WHILE first <= last
          IF numList[mid] == key
              PRINT "Element found at position", mid+1
              STOP
          ELSE IF numList[mid] > key
              last = mid - 1
          ELSE
              first = mid + 1
Step 4: PRINT "Search unsuccessful"
```

Dry Run Example:
List: `[2,3,5,7,10,11,12,17,19,23,29,31,37,41,43]`, key = 2
Iterations: mid = 7 (17), mid = 3 (7), mid = 1 (3), mid = 0 (2)
→ Key found at position 1 after 4 iterations.

3. Compare Linear Search, Binary Search, and Hashing in tabular form (minimum five points).
Answer:

Feature	Linear Search	Binary Search	Hashing
List Requirement	Unsorted/Sorted	Must be Sorted	No specific order required
Search Time	O(n)	O(log n)	O(1) ideally
Technique Used	Sequential Comparison	Divide and Conquer	Direct Access using Hash Function
Best Case	First element match	Middle element match	Direct access using hash
Worst Case	Key at last or not found	Multiple halving steps	Collision (if occurs)

4. Explain hashing and demonstrate the creation of a hash table using the remainder method with an example.
Answer:
Hashing is a technique where a hash function is used to calculate an index for each element.
Formula: `h(element) = element % size_of_hash_table`

Example:
List: `[34, 16, 2, 93, 80, 77, 51]`, table size = 10

Element	Hash Value	Index
34	4	4
16	6	6
2	2	2
93	3	3
80	0	0

Element	Hash Value	Index
77	7	7
51	1	1

Hash Table:

```
[80, 51, 2, 93, 34, None, 16, 77, None, None]
```

5. What is collision in hashing? How does it occur? Why is a perfect hash function ideal? Give one example.

Answer:

Collision occurs when two or more elements produce the same hash value and map to the same index.

Example: 16 % 10 = 6 and 26 % 10 = 6 → collision at index 6.

This causes a conflict since only one element can occupy a position.

A **perfect hash function** ensures **unique hash values** for every input, eliminating collisions.

6. Write and explain the Python program for binary search. Also show output for sample data.

Answer:

```
def binarySearch(list, key):
    first = 0
    last = len(list) - 1
    while first <= last:
        mid = (first + last)//2
        if list[mid] == key:
            return mid
        elif key > list[mid]:
            first = mid + 1
        else:
            last = mid - 1
    return -1
```

Sample Input:

```
List: [1, 3, 4, 5], key = 4 → Output: 4 is found at position 3
List: [12, 8, 3], key = 4 → Output: 4 is not found
```

7. Describe the working of binary search using the key = 17 in the list [2,3,5,7,10,11,12,17,19,23,29,31,37,41,43]. Create a table of each step.

Answer:

Iteration	First	Last	Mid	numList[mid]	Comparison
1	0	14	7	17	17 == 17

Result: Key found in 1st iteration at position 8.
Since 17 is the middle element, binary search finishes in one iteration.

8. What is the purpose of a hash function? Write a Python program to insert elements into a hash table and search using hashing.

Answer: A **hash function** calculates the index for an element in a hash table using a mathematical operation.

```
def hashFind(key, hashTable):
    if hashTable[key % 10] == key:
        return (key % 10) + 1
    return None

hashTable = [None] * 10
L = [34, 16, 2, 93, 80, 77, 51]

for i in L:
    hashTable[i % 10] = i

key = 16
position = hashFind(key, hashTable)
print("Found at position", position) if position else print("Not found")
```

Output:

```
Number 16 present at 7 position
```

9. Write and explain the Python program for linear search. Also show output for sample data.

Answer:

```
def linearSearch(list, key):
    for index in range(len(list)):
        if list[index] == key:
            return index + 1
    return None

list1 = [12, 23, 3, -45]
key = 23
position = linearSearch(list1, key)
```

```
if position is None:
    print("Number not present")
else:
    print("Number is present at position", position)
```

Output:

```
Number is present at position 2
```

Explanation:

The function compares each element until key is found and returns the 1-based position.

CHAPTER END EXERCISES

1. Using linear search determine the position of 8, 1, 99, and 44 in the list:

```
[1, -2, 32, 8, 17, 19, 42, 13, 0, 44]
```

Draw a detailed table showing values and decisions in each pass.

Answer:

For key = 8

Index	Value	Comparison
0	1	$1 \neq 8$
1	-2	$-2 \neq 8$
2	32	$32 \neq 8$
3	8	$8 = 8$

Position = 4, Comparisons = 4

For key = 1 → Found at index 0
Position = 1, Comparisons = 1

For key = 99 → Not present
Comparisons = 10 (entire list)

For key = 44

Index	Value	Comparison
…	…	…
9	44	44 = 44

Position = 10, Comparisons = 10

---

2. Use the linear search program to search key = 8 in the list:
`[42, -2, 32, 8, 17, 19, 42, 13, 8, 44]`

What is the position returned? What does this mean?

Answer: First occurrence of key = 8 is at **index 3**, so position returned is **4**.
It means linear search returns the **first match** only — it does **not check for duplicates** beyond the first match.

---

3. Write a program that takes a mix of 10 positive and negative integers and a key. Apply linear search and display if the key is found and its position. Run for 3 different keys.

Answer (sample run): List: `[-7, 3, -1, 9, -4, 8, 2, -3, 6, -9]`

Case 1: key = 8 → Found at position 6
Case 2: key = -9 → Found at position 10
Case 3: key = 5 → Not found

---

4. Write a program that takes a list of 10 integers and applies binary search.
Run the program for 3 different key values.

Answer (sample sorted list):
List: `[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]`

Case 1: key = 7 → Found at position 4
Case 2: key = 1 → Found at position 1
Case 3: key = 12 → Not found

---

5. Given the list:
`[50, 31, 21, 28, 72, 41, 73, 93, 68, 43, 45, 78, 5, 17, 97, 71, 69, 61, 88, 75, 99, 44, 55, 9]`

a) Use linear search to find positions of 1, 5, 55, 99
b) Sort the list in ascending order
c) Use linear search again
d) Use binary search and record iterations

Answer:

a) Linear Search (Unsorted List)
- 1 → Not found (24 comparisons)
- 5 → Found at position 13
- 55 → Found at position 23
- 99 → Found at position 21

b) Sorted List:
`[5, 9, 17, 21, 28, 31, 41, 43, 44, 45, 50, 55, 61, 68, 69, 71, 72, 73, 75, 78, 88, 93, 97, 99]`

c) Linear Search (Sorted List)
Same results, still sequential comparisons

d) Binary Search Iterations
- 1 → Not found in $log_2 24 \approx 5$ iterations
- 5 → Found at position 1 in ~4 iterations
- 55 → Found at position 12 in ~3 iterations
- 99 → Found at position 24 in ~4 iterations

---

6. Given the list of words:
`[Perfect, Stupendous, Wondrous, Gorgeous, Awesome, Mirthful, Fabulous, Splendid, Incredible, Outstanding, Propitious, Remarkable, Stellar, Unbelievable, Super, Amazing]`

a) Use linear search to find 'Amazing', 'Perfect', 'Great', 'Wondrous'
b) Sort list alphabetically
c) Use linear search again
d) Use binary search and note iterations

Answer:

a) Linear Search (Unsorted List): - Amazing → Found at position 16
- Perfect → Found at position 1
- Great → Not found
- Wondrous → Found at position 3

b) Sorted List:
`[Amazing, Awesome, Fabulous, Gorgeous, Incredible, Mirthful, Outstanding, Perfect, Propitious, Remarkable, Splendid, Stellar, Stupendous, Super, Unbelievable, Wondrous]`

c) Linear Search on Sorted List:
Positions change based on sorting

d) Binary Search Iterations (approx.):
- Amazing → 1 iteration
- Perfect → 3 iterations
- Great → 4 iterations → Not found
- Wondrous → 4 iterations

7. **Estimate the number of key comparisons required in binary search and linear search** if we need to find the details of a person in a **sorted database having 230 (1,073,741,824) records** when the details of the person being searched lie at the middle position in the database.
   What do you interpret from your findings?

Answer

Linear Search: - In linear search, if the element is at the **middle position**, the number of comparisons = $\lceil 230/2 \rceil = \mathbf{115}$

Binary Search: - Binary search uses $log_2(n)$ comparisons in the worst case. - So, comparisons = $\lceil log_2(230) \rceil \approx \lceil 7.85 \rceil = \mathbf{8}$

Interpretation: - Linear search requires **115 comparisons** to find the middle element. - Binary search needs only **8 comparisons** for the same.

This highlights that binary search is significantly more efficient than linear search, especially for larger datasets—even when the item is at the middle position.

8. Use hash function `h(element) = element % 11` to store:
`[44, 121, 55, 33, 110, 77, 22, 66]`
Search for 11, 44, 88, 121

Answer:

Hash Table (size 11):

Element	Hash Value
44	0
121	0 (collision)
55	0 (collision)
33	0 (collision)
110	0 (collision)
77	0 (collision)
22	0 (collision)
66	0 (collision)

All elements map to index 0 — illustrates severe **collision**.

Search Results:
- 44 → Present (if placed first)

- 121 → Could be lost due to collision
- 88 → Not present
- 11 → Not present

Requires **collision resolution**.

---

9. Given:

```
CountryCapital = {
  'India':'New Delhi', 'UK':'London', 'France':'Paris',
  'Switzerland': 'Berne', 'Australia': 'Canberra'
}
```

Use hash index = length of country name - 1.
Fill 2 lists (keys and values) and search capitals of India, France, and USA.

Answer: Hash Indexes:

Country	Length	Index
India	5	4
UK	2	1
France	6	5
Switzerland	11	10
Australia	9	8

Keys List:

```
[None, 'UK', None, None, 'India', 'France', None, None, 'Australia', None,
'Switzerland']
```

Values List:

```
[None, 'London', None, None, 'New Delhi', 'Paris', None, None, 'Canberra',
None, 'Berne']
```

Search Results:

- India → New Delhi

- France → Paris
- USA → Not present (hash index 2 → None)

For more information Visit:

https://matheenhere.blogspot.com