1 Fundamentals of algorithms

1 Fundamentals of algorithms

Learning objectives:

- *Understand and explain how the linear search algorithm works*
- *Understand and explain how the binary search algorithm works*
- *Compare and contrast linear and binary search algorithms*

1.3 Searching algorithms

Linear search

Imagine a pile of animal name playing cards placed face down on a table in no particular order. The playing cards are labelled *ant*, *bee*, *cat*, *dog*, and *fox* and the pile is arranged as shown in *Figure 1.3.1*.

Searching for a particular card, say "cat", by turning over the cards in turn, starting from the card on top, is called a **linear search**. The red arrow in *Figure 1.3.1* indicates the cards that have to be examined before the card labelled "cat" is found.

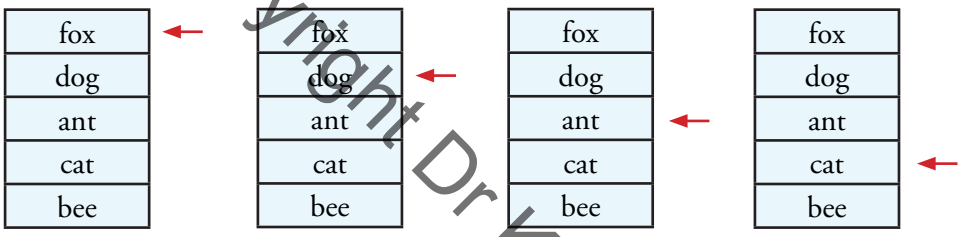

Figure 1.3.1 Linear search for the card labelled "cat"

Key point

Linear search:
Linear search scans each item or element in a collection of items, e.g. playing cards, in turn, starting from the beginning, until a match is found or the end of the collection is reached.

Linear search doesn't care whether the list is ordered or not.

Information

Linear search is often performed on lists of things, e.g. names

Questions

1. A pack of cards is shuffled to ensure that the cards are in no particular order and then placed face down on a table. Starting from the top of the pack, one playing card is turned over at a time until the Ace of Spades is found.

   (a) If the task was repeated many times, shuffling the pack of 52 cards before each new search, on average how many cards would need to be turned over to find the Ace of Spades?

   (b) What is the maximum number of cards that need turning over to find a match?

2. Approximately half of the pack is removed. Starting from the top of the pack, one playing card is turned over at a time until either the Ace of Spades is found or all the cards have been examined.
   What is the maximum number of cards that need turning over to find the Ace of Spades or to discover that the the half-pack doesn't contain the Ace of Spades?

3. What is the maximum number of cards that have to be turned over in the pile of cards in Figure 1.3.1 to discover that "rat" is not amongst them?

Key concept

Search length:
Search Length = no of elements of the vector which are examined before a match is found

Key fact

Average search length:
Average search length
$\approx \frac{\text{NoOfElementsInVector}}{2}$

Algorithm for linear search

Labelling the pile of animal name playing cards with the name, `Vector`, enables us to refer to the card on top as the card in location `Vector[1]`, the card below this card as the card in location in `Vector[2]`, and the j^{th} card as the card in location in `Vector[j]`.

1	fox
2	dog
3	ant
4	cat
5	bee

Vector

Labelling the card that we are searching for, `ElementSought`, means that we can change this card to a different one and continue to refer to the card to search for by the label `ElementSought`. The number of elements, `NoOfElementsInVector`, is 5 in our example. The algorithm below performs a linear search on `Vector` assigning to `Result` the position in `Vector` of the element if found otherwise assigning it the value 0.

Linear Search Algorithm

```
j ← 0
Found ← False
REPEAT
  j ← j + 1
  IF Vector[j] = ElementSought THEN
    Found ← True
  ENDIF
UNTIL Found Or j = NoOfElementsInVector
IF Found THEN
  Result ← j
ELSE Result ← 0
ENDIF
```


Task

1 Code the linear search algorithm in a programming language with which you are familiar. `Vector` can be implemented as a one-dimensional array of animal name strings or its equivalent. The animal name to search for should be entered at the keyboard and assigned to `ElementSought`. Your program should display the value assigned to `Result`.

Questions

4 What is meant by linear search?

Binary search

If the elements have been ordered then a much shorter average search length can be achieved as follows:

Assuming elements in a list are stored in ascending order as shown in *Figure 1.3.2*, a search for an element with a particular value, e.g. "dog", resembles the way a telephone directory might be searched.

The approximate middle of the list is located (location labelled 5 in *Figure 1.3.2*) and its value examined.

If this value is too high (e.g. alphabetically) then the approximate position of the middle element of the first half is calculated and its value examined.

If the value is too low then then the approximate position of the middle element of the second half is calculated and its value examined.

This process continues until the desired element is found or the search interval becomes empty.

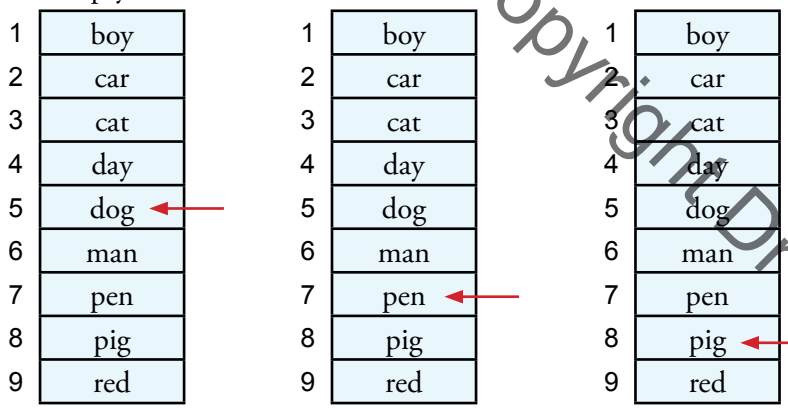

Figure 1.3.2 Performing a binary search for the word "pig" on an ordered list of words

Figures 1.3.2 and 1.3.3 show an example of binary search on an ordered list of three-letter words. The elements in the list have been numbered, 1, 2, 3, ... 7, 8, 9, for convenience. The list is searched for the word "pig" which is located at position 8 in the list.

The middle element, "dog" is selected first and compared with "pig". It doesn't match.

As "pig" is alphabetically greater than "dog", the second half of the list "boy" to "red" is chosen to search next. This second half runs from "man" to "red".

Its middle lies between the word "pen" and the word "pig". We have to choose one or the other so the word that comes first, "pen", is chosen. It doesn't match the word "pig". As "pig" is alphabetically greater than "pen", the second half of the list "man" to "red is chosen for the next search. This second half runs from "pig" to "red".

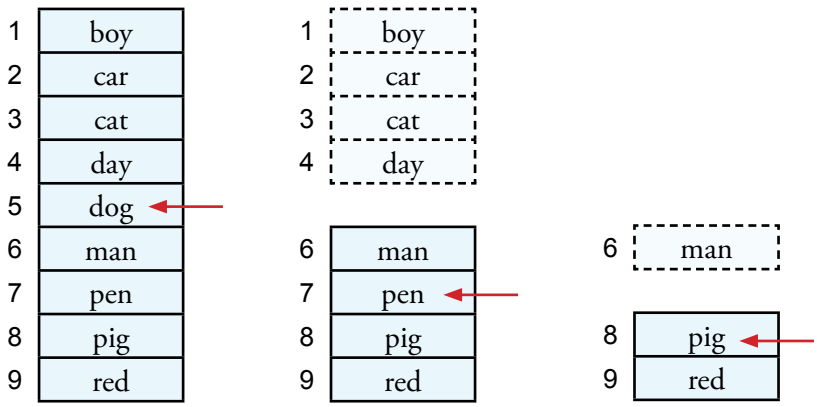

Figure 1.3.3 Performing a binary search for the word "pig" on an ordered list of words

Key point

Search interval:
The range over which the search is conducted, e.g. from list elements 1 to 9 inclusive.

Key principle

Binary search:
Searching for "pig" in the list in Figure 1.3.2 with elements labelled 1 to 9, the position of the middle element is calculated as follows

middle position = (1 + 9) div 2
= 10 div 2 = 5

middle position = (6 + 9) div 2
= 15 div 2 = 7

middle position = (8 + 9) div 2
= 17 div 2 = 8

Generalising,

middle position =
(low + high) div 2

where *low* is position no of lowest item and *high*, position no of highest item in list, e.g.

low = 8, item = pig
high = 9, item = red

Key principle

Binary search:
Binary search uses a "divide and conquer" approach to searching a list by chopping the list into smaller and smaller lists to search until item found or list cannot be divided anymore.

Its middle lies between the word "pig" and the word "red". We have to choose one or the other so the word that comes first, "pig", is chosen . It matches. So "pig" is present in the list and is located at position 8 in this list.

Questions

5 What is meant by binary search?

Algorithm for binary search

Labelling the list to be binary searched as `Vector`, enables us to refer to the first element by its location `Vector[1]`, the next element by its location `Vector[2]`, and the `j`th element by its location `Vector[j]`. The range of the vector to be searched is stored in `Low` and `High`. For example, `Low` = 1, `High` = 9 means that the beginning of the range is location `Vector`[1] and the end of the range is `Vector`[9].

Labelling the element that we are searching for, `ElementSought`, means that we can change the value to a different one and continue to refer to the element to search for by the label `ElementSought`. The algorithm below performs a binary search on `Vector` assigning to `Result` the position in `Vector` of the element if found otherwise assigning it the value -1.

Binary Search Algorithm

```
Result ← -1
WHILE (Low <= High) And (Result = -1)
  Middle ← (Low + High) Div 2  {Find middle of list}
  IF ElementSought = Vector[Middle] THEN
    Result ← Middle    {Found}
  ELSE
    IF ElementSought < Vector[Middle] THEN
      High  ← Middle - 1  {search lower half}
    ELSE
      IF ElementSought > Vector[Middle] THEN
        Low  ← Middle + 1 {search upper half}
      ENDIF
    ENDIF
  ENDIF
ENDWHILE
```

Task

2 Using the list shown in Figure 1.3.4, hand trace the binary search algorithm given above for the value "red". Complete a copy of the table shown below

Low	High	Middle
1	16	

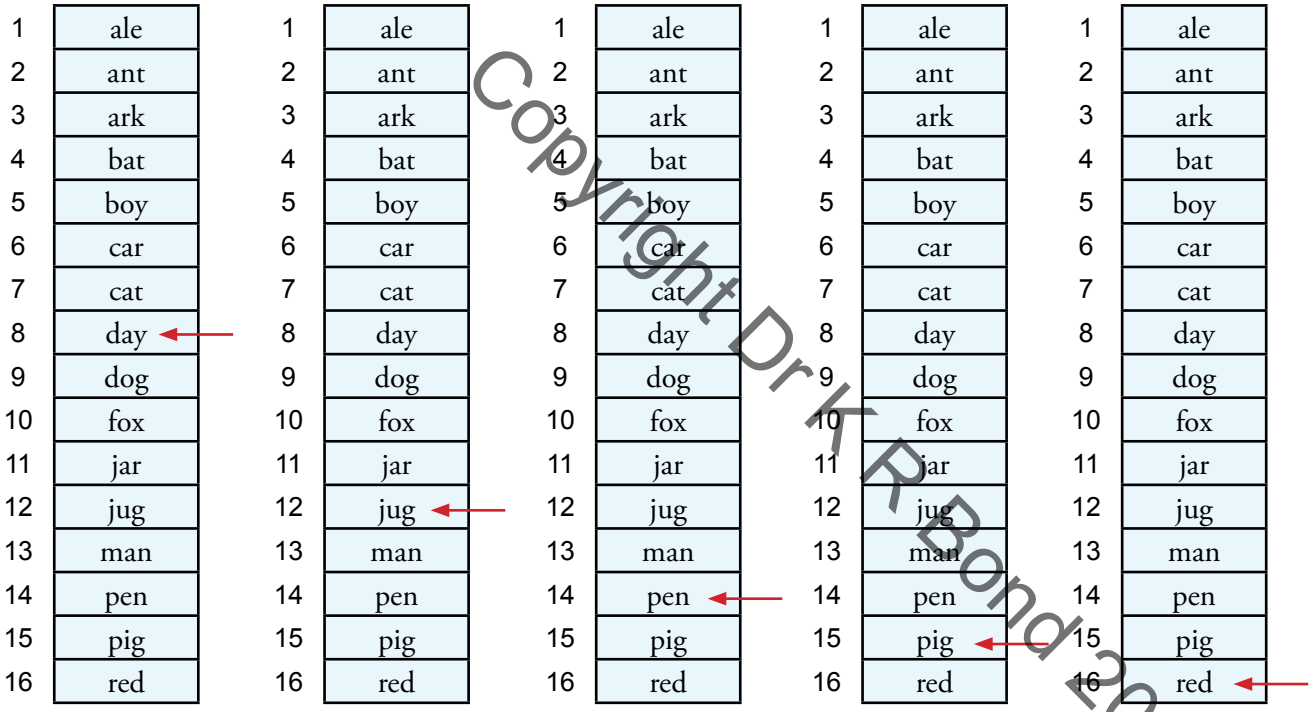

Figure 1.3.4 Performing a binary search for the word "red"on an ordered list of words

Task

3 How many elements of the list in Figure 1.3.4 have to be examined when binary searching for the element "red"?

4 How many elements have to be examined when binary searching for

(a) the element "day" in a list constructed from elements 1 to 8 of Figure 1.3.4?

(b) the element "bat" in a list constructed from elements 1 to 4 of Figure 1.3.4?

(c) the element "ant" in a list constructed from elements 1 to 2 of Figure 1.3.4?

Comparing linear and binary search algorithms

Table 1.3.1 summarises the outcomes of completing tasks 3 and 4. From *Table 1.3.1* we conclude that for binary search the maximum search length **increases linearly when the number of elements or items in a list doubles**. For example, if the number of items in the list is 8 (2^3), the maximum search length is 3 + 1, i.e. 4 items have to be examined at most to find a match or conclude that the sought item is not in the list.

If we have, say, 16777216 (2^{24}) in a list, the maximum search length is 24 + 1, i.e. 25 items have to be examined at most to find a match or conclude that the sought item is not in the list.

No of items in list	No of items in list as a power of 2	Maximum search length
1	2^0	1
2	2^1	2
4	2^2	3
8	2^3	4
16	2^4	5

Table 1.3.1 Relationship between maximum search length and no of items in list for binary search

If we contrast this with linear search, then searching a list of 8 items requires 8 items to be examined if the sought item is the last item, i.e. maximum search length for this linear search = 8.

Similarly, searching a list of 16777216 items requires 16777216 items to be examined if the sought item is the last item, i.e. maximum search length for this linear search = 16777216.

No of items in list	Maximum search length binary search	Maximum search length linear search
1	1	1
2	2	2
4	3	4
8	4	8
16	5	16
32768	16	32768
65536	17	65536
16777216	25	16777216

Table 1.3.2 Comparing maximum search length for binary and linear searches

Table 1.3.2 compares binary search with linear search for different lengths of list. This table shows clearly that **binary search is more efficient than linear search, timewise**. Each element or item of a list that has to be examined costs time. If, for argument's sake, it takes one microsecond to examine an item, then for a list of 16777216 items, binary search will take a maximum of 25 microseconds whilst linear search will take 16777216 microseconds or approximately 17 seconds.

We may draw a similar conclusion for the average search length.

Binary search can only be performed on ordered lists whereas **linear search can be performed on both ordered and unordered lists**. Sorting a list into order will take time but once ordered binary search will perform searches on the list faster than linear search will on the unordered list with the speed adavantage increasing with the size of the list.

Key fact

Binary search versus linear search:
Binary search is more efficient timewise than linear search.

Key fact

Binary search versus linear search:
Binary search can only be performed on ordered lists, linear search can be performed on both ordered and unordered lists.

Questions

6 State **two** requirements that a list must satisfy for an item to be found using binary search.

7 Explain why binary search is more efficient than linear search, timewise.

8 State whether it is possible to search an unordered list using
(a) binary search
(b) linear search

In this chapter you have covered:

- linear search algorithm scans a list from the beginning until a match is found or the end of the list is reached.
- binary search algorithm uses a "divide and conquer" approach to searching a list by chopping the list into smaller and smaller lists to search until item found or list cannot be divided anymore.
- binary search is more efficient than linear search, timewise, because it examines less elements of a list
- binary search can only be performed on ordered lists
- linear search can be performed on both ordered and unordered lists