Linear and binary search

1.2.6a · Computational thinking · revision notes

A search algorithm finds whether an item is in a list, and where. You need to know how a linear search and a binary search work, show the steps of each on a list, and compare them. Questions range from 'describe' (2 to 3 marks) to showing every stage of a binary search and choosing the better search for a situation.

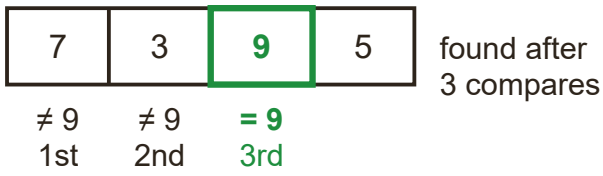
Key facts

- Linear search: check each item in turn from the start
- Binary search: the list MUST be sorted
- mid = (low + high) // 2
- Not found when low > high (no items left)
- Linear search works on unsorted lists; worst case it checks all n items
- Binary search: compare with the middle item, discard half, repeat
- Target larger than the middle item: low = mid + 1. Smaller: high = mid − 1
- Binary search maximum comparisons for n items: the smallest k where $2^k > n$ (1000 items: 10)

Notes

Linear search

- Start at the first item. Compare each item in turn with the target until it is found or you reach the end of the list.



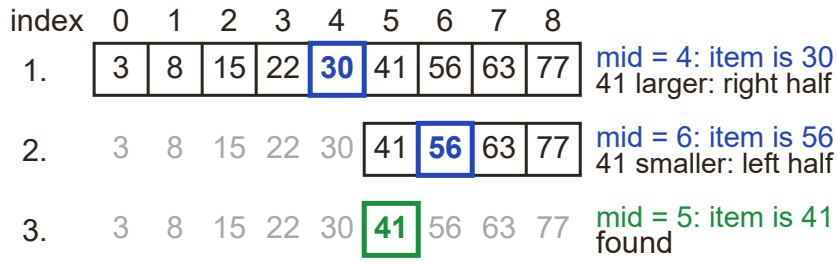
A linear search checks the items one by one from the start

- It works on **unsorted** lists as well as sorted ones.
- It is simple to write, but slow on long lists: in the worst case it checks every item.
- The flag `found` stops the loop as soon as the target is found.

```
found = False
i = 0
while i < len(items) and not found:
    if items[i] == target:
        found = True
    else:
        i = i + 1
```

Binary search

- The list must be **sorted**.
- Compare the target with the **middle** item. If they match, stop.
- If the target is smaller, discard the middle item and everything after it. If it is larger, discard the middle item and everything before it.
- Repeat with the part that is left until the item is found, or until no items are left (the item is not in the list).
- Middle position: `mid = (low + high) // 2`, where low and high are the first and last indexes of the part being searched. With an even number of items this picks the left of the two middle items. If the question gives a different rule, use that, and use the same rule every time.
- Example: find 41 in [3, 8, 15, 22, 30, 41, 56, 63, 77] (indexes 0 to 8).
  1. mid = (0 + 8) // 2 = 4: 30. 41 is larger, so search indexes 5 to 8.
  2. mid = (5 + 8) // 2 = 6: 56. 41 is smaller, so search index 5 only.
  3. mid = (5 + 5) // 2 = 5: 41. Found, after 3 comparisons.



Binary search for 41: each compare discards about half of the items that are left (greyed out)

Which search?

- Binary search is much faster on large sorted lists, because each comparison halves the number of items left: 1000 items need at most 10 comparisons, compared with up to 1000 for a linear search.
- Linear search is the better choice when the list is not sorted (sorting it first takes time) or is very short.

How to answer each type of question

Describe how a linear search works 2 to 3 marks **Grade 4**

1. Start at the first item.
2. Compare each item with the target in turn.
3. Stop when it is found, or when the end is reached (not found).

Example. Describe how a linear search would find a name in a list of names.

Model answer. Start with the first name in the list (1). Compare each name in turn with the name being searched for (1). Stop when the name is found, or report that it is not in the list when the end of the list is reached (1).

Show the stages of a binary search

3 to 4 marks **Grade 6**

1. Find the middle item and compare it with the target.
2. Say which half is discarded and write down the items that are left.
3. Repeat, showing every middle item, until the target is found or no items are left.

Example. Show the stages of a binary search to find 27 in this list. Use `(low + high) // 2` to find the middle item, where low and high are the first and last indexes of the part being searched.
[2, 5, 9, 12, 18, 21, 27, 33]

Model answer.



The middle item (index 3) is 12. 27 is greater, so 2, 5, 9 and 12 are discarded, leaving 18, 21, 27, 33 (1).
The middle item (index 5) is 21. 27 is greater, so 18 and 21 are discarded, leaving 27, 33 (1).
The middle item (index 6) is 27 (1). It matches, so the item has been found (1).

Explain which search is better

2 to 4 marks **Grade 7**

1. Check whether the list is sorted: if not, a binary search cannot be used unless the list is sorted first.
2. Compare the number of comparisons for this size of list.
3. Give reasons linked to the scenario.

Example. A shop stores 2000 product codes in ascending order.
Explain why a binary search is more efficient than a linear search for finding a product code in this list.

Model answer. Each comparison in a binary search discards half of the remaining codes (1), so it needs at most 11 comparisons, while a linear search may need to check all 2000 codes (1).

Write a linear search

4 to 6 marks **Grade 8**

1. Set up a flag (`found = False`) and a loop through the list.
2. Compare each item with the target.
3. Stop when it is found, and give an output for both found and not found.

Example. The list `names` holds the names of the members of a club. Write Python code that asks the user for a name and uses a linear search to output `Member` if the name is in the list, or `Not a member` if it is not.

Model answer.

```
target = input("Enter name: ")
found = False
i = 0
while i < len(names) and not found:
    if names[i] == target:
        found = True
    i = i + 1
if found:
    print("Member")
else:
    print("Not a member")
```

Inputs the name (1); loops through the items in the list (1); compares each item with the input (1); sets a flag or stops when the name is found (1); outputs Member if it is found (1); outputs Not a member only after the whole list has been checked (1).

Using `if target in names:` would not show the linear search that the question asks for.

Don’t lose marks

- Using a binary search on an unsorted list, or forgetting to say that the list must be sorted.
- Keeping the middle item in the part that is searched next. It has already been checked, so discard it.
- Choosing the middle item inconsistently when there is an even number of items.
- Saying that a linear search is always slower. If the item is first in the list, a linear search finds it with one comparison.
- Describing only the first comparison of a binary search. Say that the process repeats until the item is found or no items are left.

More tips

Memory tricks

- Binary search is 'higher or lower?': like guessing a number from 1 to 100 by always guessing the middle.
- Powers of 2 give the maximum number of binary search checks: $2^{10} = 1024$, so a list of up to 1023 items needs at most 10 checks.
- Linear = in a line: one after another, from the start.
- No sorted list, no binary search.

Exam technique

- When you show a binary search, write the middle item at each stage and the items that are left. Each stage is usually worth a mark.
- In 'compare' or 'explain' questions, mention sorted data and the number of comparisons, with numbers if you can.
- Use the key words: middle item, discard, half, sorted, repeat, until found.

What each grade needs

What you need to be able to do, from the first marks up to the top grade.

Grade 3 — **Describe how a linear search works**
Check each item in turn from the start until the item is found or the list ends.

Grade 4 — **State that binary search needs sorted data**
A binary search only works if the list is in order.

Grade 5 — **Describe how a binary search works**
Compare with the middle item and discard the half that cannot contain the target.

Grade 6 — **Show the stages of a binary search**
Give each middle item checked and the part of the list that is left.

Grade 7 — **Compare linear and binary search**
Weigh speed on large lists against simplicity and working on unsorted data.

Grade 8 — **Write code for a linear search**
Use a loop, a comparison and a flag that stops the search when the item is found.

Free notes, Full packs, practice papers and mock exams at chhetriacademy.com. Written for this site; not taken from past papers. Stuck? 1-to-1 tutoring with a free 30-minute trial lesson.