Efficiency of algorithms

1.2.7 · Computational thinking · revision notes

Two algorithms can both solve a problem, but one may do it with fewer steps or less memory. You need to use logical reasoning and test data to decide whether an algorithm is fit for purpose, and to judge its efficiency by the number of compares, the number of passes through a loop and its use of memory.

Key facts

- Efficiency measures: number of compares, number of passes through a loop, use of memory
- Fit for purpose: gives correct results for all valid inputs and handles invalid ones
- Test data: normal, boundary, erroneous
- Linear search: best case 1 compare, worst case n compares
- Binary search: at most k compares, where 2^k is the first power of 2 greater than n
- Bubble sort: up to n − 1 compares per pass
- Merge sort: fewer compares on large lists, but more memory
- Powers of 2: 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024

Notes

Fit for purpose

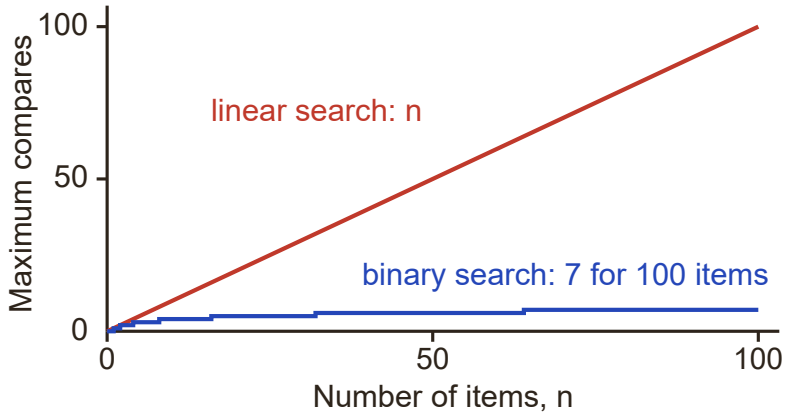
- An algorithm is **fit for purpose** if it does the job it was designed for: it gives correct results for every valid input and deals sensibly with invalid input.
- Check it with **test data**: normal data, boundary data (at the edges of the valid range) and erroneous data (data that should be rejected). Compare the expected result with the actual result.
- Use logical reasoning to think of awkward cases: the target is the first or last item or not in the list at all, all the values are negative, or two values are equal.

Measuring efficiency

- **Number of compares**: how many comparisons are made. Fewer compares means less time.
- **Number of passes through a loop**: every pass takes time, so stop a loop as soon as the answer is known, e.g. with a flag that is set when the item is found.
- **Use of memory**: extra variables, lists and copies of data all take up memory. An algorithm that needs less memory is more efficient in this respect.
- Other ways to save work: do a calculation once before a loop instead of on every pass, and don't check items that are already known to be in place.

The standard algorithms compared

- Linear search of n items: best case 1 compare (the first item), worst case n compares.
- Binary search of n sorted items: at most k compares, where 2^k is the first power of 2 greater than n. For 100 items, $2^7 = 128$, so at most 7 compares.



Worst-case compares: linear search grows with every item; binary search grows by only 1 each time the list doubles

- Bubble sort: up to n − 1 compares per pass. If it stops after a pass with no swaps, an already sorted list needs only one pass. It needs almost no extra memory.
- Merge sort: far fewer compares than a bubble sort on large lists, but it needs extra memory to hold the sublists.
- So there can be a trade-off between time (compares and passes) and memory.

How to answer each type of question

State the number of compares or passes 1 to 2 marks **Grade 6**

1. Linear search: the worst case is the number of items.
2. Binary search: find the first power of 2 that is greater than the number of items.
3. Bubble sort that compares every pair on each pass: compares = passes × (n − 1).

Example. (a) A sorted list contains 60 items. State the maximum number of compares a linear search would make to find an item.
(b) State the maximum number of compares a binary search would make to find an item in the same list.
(c) A bubble sort of a list of 6 items takes 4 passes. Each pass compares every adjacent pair. State the total number of compares.

Model answer.

(a) 60 (1)

(b) 6 (1), because $2^6 = 64$, which is greater than 60

(c) 4 × 5 = 20 (1)

Explain how to make an algorithm more efficient

2 to 3 marks **Grade 7**

1. Find the wasted work: extra passes, extra compares or extra memory.
2. Say what to change.
3. Say what is saved: fewer compares, fewer passes or less memory.

Example. This algorithm checks whether `target` is in the list `names`.

```
found = False
for i in range(len(names)):
    if names[i] == target:
        found = True
print(found)
```

Explain how the algorithm could be made more efficient.

Model answer. Stop the loop as soon as the target is found, e.g. use a while loop that ends when found is True (1). No more items are then compared after the target has been found (1), so there are fewer passes through the loop when the target is near the start of the list (1).

Explain why an algorithm is not fit for purpose

3 to 4 marks **Grade 7**

1. Find input data for which the algorithm gives the wrong result.
2. Give the expected result and the actual result.
3. Explain the cause and give a correction.

Example. This algorithm should output the highest temperature in the list `temps`.

```
largest = 0
for t in temps:
    if t > largest:
        largest = t
print(largest)
```

Using suitable test data, explain why the algorithm is not fit for purpose, and state how to correct it.

Model answer. Test data: a list in which every temperature is below zero, e.g. [−3, −7, −1] (1). The expected output is −1, but the algorithm outputs 0 (1), because largest starts at 0, which is higher than every temperature, so it is never replaced (1). Correct it by starting with the first item: `largest = temps[0]` (1).

Evaluate the efficiency of two algorithms

4 to 6 marks | **Grade 8**

1. Compare the number of compares (or passes) for this amount of data.
2. Compare the memory each one needs.
3. Consider other requirements, such as whether the data is sorted.
4. Make a justified choice for the scenario.

Example. A library app searches a list of 500 book titles. The list is sorted every night and is searched thousands of times a day.
Evaluate the use of a linear search and a binary search for this app.

Model answer. A linear search may need to compare all 500 titles (1), but a binary search needs at most 9 compares, because each compare halves the titles that are left (1). A binary search needs the list to be sorted (1); this list is sorted every night, so that is possible (1). Neither search needs much extra memory, so time is the deciding factor (1). As the list is searched thousands of times a day, the binary search is the better choice (1).

Don't lose marks

- Saying an algorithm is 'more efficient' or 'faster' without saying why: fewer compares, fewer passes or less memory.
- Giving n ÷ 2 as the maximum number of compares for a binary search. It is far smaller: roughly the number of times n can be halved.
- Forgetting the final pass with no swaps when counting the passes of a bubble sort that stops when a pass makes no swaps.
- Testing only with normal data, which can hide the cases where an algorithm fails.

More tips

Memory tricks

- Learn the powers of 2 up to 1024: the first one bigger than n gives the binary search maximum.
- Bubble sort that compares every pair on each pass: compares = passes × (n − 1).
- Time v memory: merge sort is fast but uses more memory; bubble sort is slow but uses very little.
- To test 'fit for purpose', try the awkward cases: first item, last item, missing item, negative numbers, equal values.

Exam technique

- Use numbers from the question: 'at most 9 compares instead of 500' is stronger than 'much faster'.
- Mention best and worst cases where they differ, e.g. a linear search finds the first item with one compare.
- In 'evaluate' questions, cover time and memory, link each point to the scenario, and finish with a judgement.

What each grade needs

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

Grade 4

Count comparisons in a linear search

Count how many items are checked before the target is found or the list ends.

Grade 5

Use test data to check an algorithm

Compare the expected result with the actual result for normal, boundary and erroneous data.

Grade 5

Count passes and comparisons in a bubble sort

In the basic version, each pass through a list of n items makes n − 1 comparisons.

Grade 6

Find maximum comparisons for a binary search

Find the smallest power of 2 that is greater than the number of items.

Grade 7

Suggest how to make an algorithm more efficient

E.g. stop a loop as soon as the answer is known.

Grade 7

Explain why an algorithm is not fit for purpose

Use test data to show a case where it gives the wrong result.

Grade 8

Evaluate algorithms by time and memory

Judge which algorithm suits a scenario, using compares, passes and memory use.

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.