Sequential search

```
procedure Search(List, TargetValue):
{ Input: List is a list; TargetValue is a possible entry }
{ Output: success if TargetValue in List; failure otherwise }

    if (List empty)
        then Output failure

        else
            TestEntry := 1st entry in List
            while (TargetValue ≠ TestEntry
                    and there are entries not considered)
                (TestEntry := next entry in List)
            if (TargetValue = TestEntry)
                then Output success
                else Output failure
```

Sequential search

Analysis:

- time
- fundamental operation
  - takes time
  - number of occurrences proportional to everything else that happens

Sequential search

Analysis:

$|\text{List}| = n$

How many comparisons are necessary in the worst case?

A. 1
B. $n - 1$
C. n
D. $n + 1$
E. $2n$

Vote at m.socrative.com. Room number 415439.

Sequential search

Analysis:

$|\text{ List }| = n$

How many comparisons are necessary in the worst case?

D. $n + 1$

This is $\Theta(n)$.

Sequential search

Analysis:

What does $\Theta(n)$ meant?

Need to define $O(n)$ too.

$g \in O(f)$ means $\exists c, d$ s.t. $g(n) \leq c \cdot f(n) + d$

$g \in \Theta(f)$ means $g \in O(f)$ and $f \in O(g)$.

Sequential search

Analysis:

$g \in O(f)$ means $\exists c, d$ s.t. $g(n) \leq c \cdot f(n) + d$

$g \in \Theta(f)$ means $g \in O(f)$ and $f \in O(g)$.

Examples:

- $2n + 3 \in \Theta(n)$
- $3 \log n \in \Theta(\log n)$
- $2n + 7 \log n \in \Theta(n)$
- $4 \log n + m \in \Theta(\log n)$ if $m \leq \log n$
- Can write $\Theta(\log n + m)$ if unsure which term is larger.

Sequential search

Analysis:

What is $n \log n - 1.4n + 15$?

A. $O(n^2)$
B. $O(n \log n)$
C. $\Theta(n \log n)$
D. all of the above
E. none of the above

Vote at m.socrative.com. Room number 415439.

Sequential search — correctness

```
procedure Search(List, TargetValue):
    if (List empty)
        then Output failure
        else
            TestEntry := 1st entry in List
        { precondition: TestEntry is 1st entry in List }
            while (TargetValue ≠ TestEntry
                        and there are entries not considered)
                  (TestEntry := next entry in List)
        { loop invariant: TargetValue ≠ any entry before TestEntry }
        { postcondition: either TargetValue = TestEntry
            or all entries considered and TargetValue not in List }
            if (TargetValue = TestEntry)
                then Output success
                else Output failure
```

Sequential search — correctness

Assertions

- statements which can be proven to hold (induction)
- at different points in program
- examples: precondition, postcondition, loop invariant

Proof by induction on number of times through the loop:

Proof verification: automated?

Sequential search — correctness

Searching a sorted list

7	8	15	53	54	61	66	75	77	99	104	111	123	124	150
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Find 104. How many comparisons with sequential search?

A. 1
B. 4
C. 11
D. 12
E. 16

Vote at m.socrative.com. Room number 415439.

Sorted list

D. 12

Can we do better?

Binary search

```
procedure Search(List, TargetValue):
{ Input: List is a list; TargetValue is a possible entry }
{ Output: success if TargetValue in List; failure otherwise }

    if (List empty)
        then Output failure

        else
            TestEntry = middle entry in List
            if (TargetValue = TestEntry)
                then Output success
                else if (TargetValue < TestEntry
                    then Search(left-of-TestEntry, TargetValue)
                    else Search(right-of-TestEntry, TargetValue)
```

Binary search

Recursion

- contains reference to itself (subtask)
- termination condition (no infinite loops) — base case

Binary search

7	8	15	53	54	61	66	75	77	99	104	111	123	124	150
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TargetValue: 104

Middle index: 8

TestEntry: 75

Binary search

```
procedure Search(List, TargetValue):
{ Input: List is a list; TargetValue is a possible entry }
{ Output: success if TargetValue in List; failure otherwise }

    if (List empty)
        then Output failure

        else
            TestEntry := middle entry in List
            if (TargetValue = TestEntry)
                then Output success
                else if (TargetValue < TestEntry
                    then Search(left-of-TestEntry, TargetValue)
                    else Search(right-of-TestEntry, TargetValue)
```

Binary search

7	8	15	53	54	61	66	75	77	99	104	111	123	124	150
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TargetValue: 104

Middle index: 12

TestEntry: 111

Binary search

```
procedure Search(List, TargetValue):
{ Input: List is a list; TargetValue is a possible entry }
{ Output: success if TargetValue in List; failure otherwise }

    if (List empty)
        then Output failure

        else
            TestEntry := middle entry in List
            if (TargetValue = TestEntry)
                then Output success
                else if (TargetValue < TestEntry
                    then Search(left-of-TestEntry, TargetValue)
                    else Search(right-of-TestEntry, TargetValue)
```

Binary search

7	8	15	53	54	61	66	75	77	99	104	111	123	124	150
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TargetValue: 104

Middle index: 10

TestEntry: 99

Binary search

```
procedure Search(List, TargetValue):
{ Input: List is a list; TargetValue is a possible entry }
{ Output: success if TargetValue in List; failure otherwise }

    if (List empty)
        then Output failure

        else
            TestEntry := middle entry in List
            if (TargetValue = TestEntry)
                then Output success
                else if (TargetValue < TestEntry
                    then Search(left-of-TestEntry, TargetValue)
                    else Search(right-of-TestEntry, TargetValue)
```

Binary search

7	8	15	53	54	61	66	75	77	99	104	111	123	124	150
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TargetValue: 104

Middle index: 11

TestEntry: 104

Binary search

```
procedure Search(List, TargetValue):
{ Input: List is a list; TargetValue is a possible entry }
{ Output: success if TargetValue in List; failure otherwise }

    if (List empty)
        then Output failure

        else
            TestEntry := middle entry in List
            if (TargetValue = TestEntry)
                then Output success
                else if (TargetValue < TestEntry
                    then Search(left-of-TestEntry, TargetValue)
                    else Search(right-of-TestEntry, TargetValue)
```

Binary search

7	8	15	53	54	61	66	75	77	99	104	111	123	124	150
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TargetValue: 104

Result: success

Binary search

Recursion

- contains reference to itself (subtask)
- termination condition (no infinite loops) — base case
- need:
  - initialization
  - modification
  - test for termination
- no more powerful than iteration, but easier to program

Divide-and-Conquer — algorithmic technique

reduce to smaller problem(s)

Binary search — analysis

Each list has length $\leq \frac{1}{2}$ the previous.

List sizes: $n, \ \lfloor \frac{n}{2} \rfloor, \ \lfloor \frac{n}{4} \rfloor, \ \lfloor \frac{n}{8} \rfloor, \ldots, 1$

1 comparison per list size.

Worst case: $1 + \lfloor \log_2 n \rfloor$ comparisons — $\Theta(\log(n))$

Can it take this many comparisons?

Binary search — analysis

Each list has length $\leq \frac{1}{2}$ the previous.

List sizes: $n, \ \lfloor \frac{n}{2} \rfloor, \ \lfloor \frac{n}{4} \rfloor, \ \lfloor \frac{n}{8} \rfloor, \ldots, 1$

1 comparison per list size.

Worst case: $1 + \lfloor \log_2 n \rfloor$ comparisons — $\Theta(\log(n))$

Can it take this many comparisons?

Yes.

Binary search — uses

Binary search can be used in many situations.

There does not need to be an explicit list.

In an implicit list, one could have functions of the index, such as $f(n) = (n+1)^2$ or $f(n) = 2^n$.

Sorting

How do you sort? Think about cards.

Insertion Sort

```
procedure Sort(List):
{ Input: List is a list }
{ Output: List, with same entries, but in nondecreasing order }

    N := 2
    while (N ≤ length(List)
    begin
        Pivot := Nth entry
        j := N − 1
        while (j > 0 and jth entry > Pivot)
        begin
            move jth entry to loc. j + 1
            j := j − 1
        end
        place Pivot in j + 1st loc.
        N := N + 1
    end
```