Searching

Dov Kruger

Department of Electrical and Computer Engineering
Rutgers University

March 4, 2024

Introduction

Searching happens more than sorting
Often worth the expense to sort for later high speed searching

- Linear Search
- Binary Search
- Golden Mean Search

Linear search has two possible outcomes

Target is found (must be between 0 and n-1)

Target is not found O(n)

If the target is found, and the element is random the position will vary from 0 to n-1

Assuming the selection is equally likely position is O(n/2) which is O(n).

Consider the following cases:

- Search for 31
- Search for 2
- Search for 5000

55	2	46	86	21	72	-11	4	...	31

Linear Search Pseudocode

Q: What is the complexity?

```
linearSearch(a, target)
  for i ← 0 to length(a)
    if a[i] = target
      return i
    end
  end
  return -1 // not found
end
```

Linear Search of a Sorted List

For a sorted list, is searching any faster?

Example: Search for the number 95

Example: Search for the number 2

Example: Search for the number 25

-11	2	4	27	39	41	47	56	...	95

Linear Search Sorted Pseudocode

Q: What is the complexity?

```
linearSearch(a, target)
  for i ← 0 to length(a)
    if a[i] = target
      return i
    else if a[i] > target
      return −1
    end
  end
  return −1 // not found
end
```

Binary Search requires a sorted list

Much faster O(log n) time once the search is done.
The cost to sort is $O(n \log n)$ so it's a very small cost overall if we consider the number of searches and the applications of searches

Let's do this interactively

```
binarySearch(a, target)
  binarySearch(a, target, 0, length(a)−1)
end

binarySearch(a, target, L, R)
  if L > R
    return −1  // can't find value if there are no
  end
  mid ← (L + R)/2
  if a[mid] > target
    return binarySearch(a, target, mid + 1, R)
  else if a[mid] < target
    return binarySearch(a, target, L, mid−1)
  return mid
end
```

With a slightly wrong algorithm, binary search will never terminate

```
binarySearch(a, target, L, R)
  if L ≥ R
    return −1  // can't find value if there are no
  end
  mid ← (L+R)/2
  if a[mid] > target
    return binarySearch(a, target, mid, R) // wrong
  else if a[mid] < target
    return binarySearch(a, target, L, mid) // works
  return mid
end
```

Worst Case

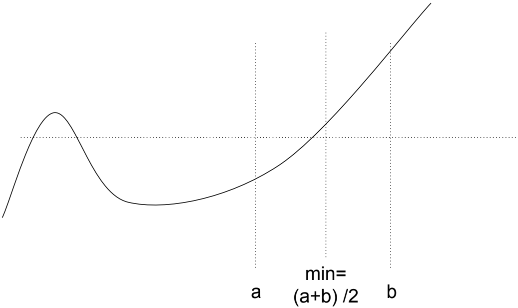
Assume the list is - 1 2 3 4 6 and we need to find 5
The mid element at the first pass $= 3$
At the second pass, mid $= 4$
Now, due to the worst case, we have the position of mid $=$ $(3+4)/2 = 3$
So we end up with a loop that never terminates

Continuous Space: Bisection

The bisection algorithm is the continuous version of binary search
Used to find the roots of functions
Assumptions:

- Function is continuous
- One side is negative, the other positive
- The function, therefore, goes through zero (has a root on the interval)

Bisection



Bisection Algorithm Pseudocode

```
bisection(f, a, b, tolerance, iterations)
    i = 1
    while i ≤ iterations
        mid = (a + b) / 2
        if f(mid) = 0 OR (b − a) / 2 < tolerance
            return mid
        i++
        if f(a) * f(mid) < 0
            b = mid
        else
            a = mid
    end
    return "Maximum␣Steps␣Crossed"
end
```

Golden Mean Search

Golden mean is a way of optimizing for max and min, given that the exact value is not known.
For the purposes of discussion, we will consider only the maximum since it's the same.
Assumptions:

- The function has a single global maximum
- The function does not have any other local maxima

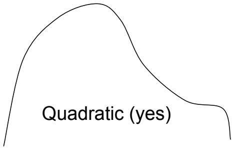
In discrete space, we are looking for the maximum value of a list
In continuous space, we are looking for the maximum value of the function

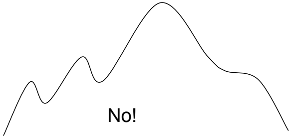
Golden Mean Example: List

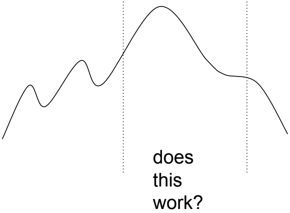
n=20

-40	-38	22	23	29	29	29	37	55	56	57	57	61	92	32	12	10	2	1	0

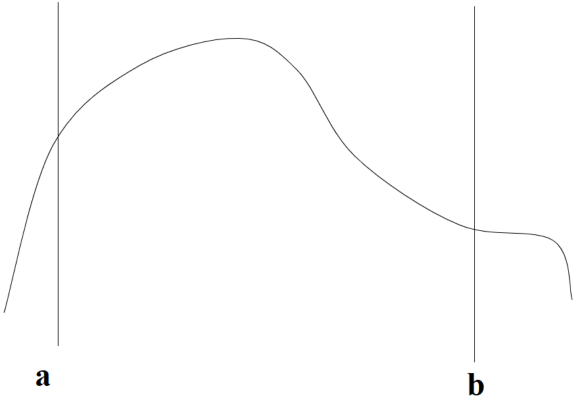
Golden Mean Example: Function





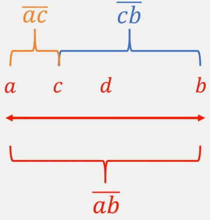


Golden Mean



Pick 2 points between a and b, say c and d to minimize search space to either a to d or b to c

- What happens if c and d are arbitrary?
- Can we find c and d such that the new search space remains the same irrespective of which partition to choose?



$$\frac{\overline{ab}}{\overline{cb}} = \frac{\overline{cb}}{\overline{ac}}$$

$$\frac{\overline{ab}}{\overline{cb}} = \frac{\overline{cb} + \overline{ac}}{\overline{cb}} = \frac{\overline{cb}}{\overline{cb}} + \frac{\overline{ac}}{\overline{cb}} = 1 + \frac{\overline{ac}}{\overline{cb}}$$

$$\left(\frac{\overline{cb}}{\overline{ac}}\right)\frac{\overline{cb}}{\overline{ac}} = \left(1 + \frac{\overline{ac}}{\overline{cb}}\right)\left(\frac{\overline{cb}}{\overline{ac}}\right)$$

$$\left(\frac{\overline{cb}}{\overline{ac}}\right)^2 = \frac{\overline{cb}}{\overline{ac}} + 1$$

$$\left(\frac{\overline{cb}}{\overline{ac}}\right)^2 - \frac{\overline{cb}}{\overline{ac}} - 1 = 0$$

$$\frac{\overline{cb}}{\overline{ac}} = \phi = \frac{1+\sqrt{5}}{2} \approx 1.618$$

The magic constant $\phi = (1+\sqrt{5})/2$

Golden Mean Search Algorithm

```
GoldenMean(func, a, b)
s ← (b − a)/φ
d ← a + s
c ← b − s
if func(c) > func(d)
   b ← d
   d ← c
   s ← (b − a)/φ
   c ← b − s
else
   a ← c
   c ← d
   s ← (b − a)/φ
   d ← a + s
end
```

Let's do this interactively

Why is it the Golden Ratio?

Reduction of search space by $1/\phi$
$\phi = 1.618$
But what is $1/\phi? = 0.618$
$1 - \phi = 0.618$

Golden Mean Interactive demonstration