Interpolation Search and *MaxMin*

Textbook Reading:

- Subsection 2.6.3, pp. 38-39
- Subsection 2.6.4, pp. 39-41

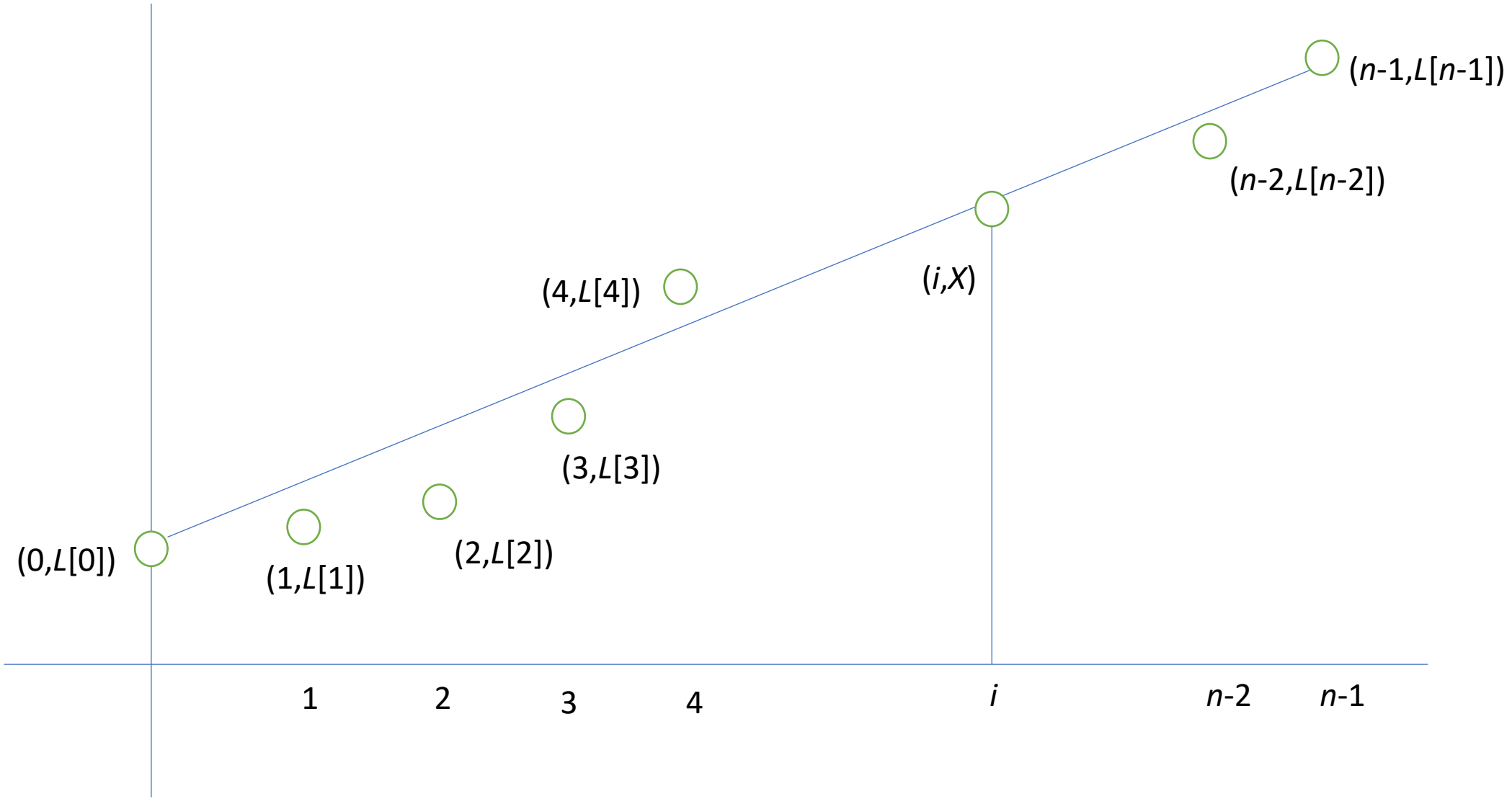
Interpolation Search

In Interpolation Search, like in Binary Search, we make the assumption that the values in the original list $L[0:n-1]$ are not only increasing, but lie approximately along a straight line joining the points $(0,L[0])$ to $(n-1,L[n-1])$.

Interpolation Search

Interpolation Search is the same as Binary Search except instead of computing the midpoint it computes the index *i* such that (*i*,*X*) lies on the line joining (*low*,*L*[*low*]) to (*high*,*L*[*high*])

Interpolation Search



Recursive Implementation

- To implement the recursion as usual we must introduce *low* and *high* and perform the interpolation search on the sublist *L*[*low*,*high*].
- In this more general version, the line contains the three points:

(low,$L[low]$), (i,X), ($high$,$L[high]$).

PSN. Obtain a formula for i in terms of X, low, $high$, $L[low]$, $L[high]$

PSN. Write pseudocode for recursive version of Interpolation Search.

Worst-case Complexity Analysis

Consider the list $L[0{:}n-1]$ where $L[0] = 0$ and $L[i] = 1$, $i = 1, 2, \ldots, n-1$.

Take the search element X to be an element slightly less than 1, i.e., to be $1 - \varepsilon$, where $0 < \varepsilon < \frac{1}{n}$. Then, the number of comparisons performed by *InterpolationSearch* is $n + 2$, *i.e.,*

$$W(n) = n + 2.$$

Proof. *Applying the formula mid with low = 0, high = n,* $L[0] = 0$, $L[n] = 1$, $X = 1 - \varepsilon$, *we obtain*

$$mid = \lfloor low + (X - L[low])(high - low)/(L(high) - L(low)) \rfloor$$

$$= \lfloor (n-1)(X - L[0])/(L(n-1) - L(0) \rfloor$$

$$= \lfloor (n-1)(1-\varepsilon) \rfloor = n - 2.$$

Thus, after performing the two comparisons $X = L[n-2]$ *and* $X < L[n-2]$, *InterpolationSearch is recursively called with the list* $L[0{:}n-3]$, so that the sublist we need to search has only been reduced by two elements. *After performing two more comparisons InterpolationSearch is called with the list* $L[0{:}n-5]$. This continues until we have reached the bootstrap condition. Altogether n comparisons are performed. Adding on the two comparisons to check whether $L[low] \leq X \leq L[high]$, *we have a total of* $n + 2$ *comparisons.*

Poor performance in worst-case but very fast on average

- We have shown that Interpolation Search makes about n comparisons in the worst-case. This is very poor performance. It's worse than Linear Search, which works without the assumption that the list is sorted.
- However, under suitable assumptions of randomness for the elements of the list $L[0{:}n-1]$, it can be shown that the average performance $A(n)$ of interpolation search is approximately $\log_2(\log_2 n)$, a very slowing growing function indeed.

Computing the maximum element in as list

```
function Max (L[0:n – 1])
Input:  L[0:n – 1] (a list of size n)
Output:          returns the maximum value occurring in L[0:n – 1]
  MaxValue ← L[0]
  for i ← 1 to n – 1  do
        if L[i] > MaxValue then
              MaxValue ← L[i]                    //update MaxValue
        endif
  endfor
  return(MaxValue)
end Max
```

An algorithm *Min* for computing the minimum element in a lists can be coded similarly.

Analysis of *Max* and *Min*

$$B(n) = A(n) = W(n) = \ n - 1.$$

Computing max and min elements in a list

What about simply calling functions *Max* and *Min* for computing the maximum and minimum elements in a list respectively?

$$W(n) = 2n - 2.$$

When can do 25% better, i.e., we can achieve

$$W(n) = (3/2)n - 2.$$

Better Algorithm

procedure *MaxMin2*(*L*[0:*n* – 1],*MaxValue*,*MinValue*)
Input: *L*[0:*n* – 1] (a list of size *n*)
Output: *MaxValue*,*MinValue* (maximum and minimum values occurring in *L*[0:*n* – 1])

```
    MaxValue ← L[0]
    MinValue ← L[0]
    for i ← 1 to n – 1  do
            if L[i] > MaxValue then
                    MaxValue ← L[i]
            else
                if L[i] < MinValue then
                        MinValue ← L[i]
                endif
            endif
    endfor
end MaxMin2
```

Analysis of *MaxMin2*

B(n) = $n - 1$. Why?

Worst-case $W(n)$ for *MaxMin2* is still $2n - 2$.

Why?

What about average complexity of *MaxMin2* assuming a uniform distribution?

A(n) = 2*n* – 2 – average number of times *MaxValue* is updated.

This average equals the number of times the maximum element is updated in Max() for a uniform distribution. It turns out that this is a value between ln n and 1 + ln n, so that its about identically equally to ln n. This was shown to be true empirally in a homework assigned given in solved in a homework assignment in Models 1 (ENED 1090):

Average Number of Updates of Max Element

It turns out that the average number of updates of the maximum element is equal to the harmonic series $H(n) = 1 + \frac{1}{2} + \ldots + 1/n$ and $\ln n < H(n) < 1 + \ln n$. We will prove this result later in this course.

It follows that

$$A(n) \approx 2n - 2 - \ln n.$$

Optimal algorithm *MaxMin3*

Strategy. Pair up and compare adjacent elements. If larger element (winner) is greater than the maximum so far, then update the maximum. If smaller element (loser) is smaller than the minimum so far, then update the minimum.

Illustration with sample list: 64 67 69 3 11 2 17 10

64 – 67 69 – 31 9 – 2 17 – 10 (involves $n/2$ comparisons in general)

Call to Max

Winners: 67 69 9 17 → 69 (involves at most $n/2 - 1$ comparisons in general)

Call to Min

Losers: 64 31 2 10 → 2 (involves at most $n/2 - 1$ comparisons in general)

Pseudocode for *MaxMin3*

```
procedure MaxMin3(L[0:n – 1],MaxValue,MinValue)
Input: L[0:n – 1] (a list of size n)
Output:     MaxValue,MinValue (the maximum and minimum values in L[0:n – 1])
    if even(n) then  //n is even
            MM(L[0],L[1],MaxValue,MinValue)
            for i ← 2 to n –  2 by 2 do
                    MM(L[i],L[i + 1],b,a)
                    if a < MinValue then MinValue ← a endif
                    if b > MaxValue then MaxValue ← b endif
            endfor
    else //n is odd
            MaxValue ← L[0]; MinValue ← L[0];
            for i ← 1 to n – 2 by 2 do
                    MM(L[i],L[i + 1],b,a)
                    if a < MinValue then MinValue ← a endif
                    if b > MaxValue then MaxValue ← b endif
            endfor
    endif
end MaxMin3
```

Analysis of *MaxMin3*

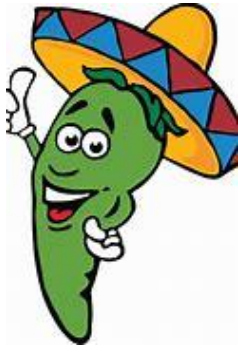
$$B(n) = A(n) = W(n) = \lceil 3n/2 \rceil - 2.$$

What do you call a bagel with the most jalapenos?

Maximum.

What do you call a bagel with just 1 jalapeno?

Minimum.



What do you call a bagel with no jalapenos?

Answer:

Optimum.