Searching

Stuart Allen

CM1103

School of Computer Science & Informatics
Cardiff University

Search

- Suppose we wish to find out if *and* where an object (that we call the *key*) appears within a sequence of objects
- E.g. we want to find whether a given credit card number is in a list of stolen credit cards
- There are several algorithms to achieve this – how do we compare them and decide which to use?
- We will contrast *linear* and *binary* search

Linear search: Example

Given a list `[ 5, 2, 8, 9 ]`

Find key of 8:

```
i= 0   a[i] = 5 ≠ 8
i= 1   a[i] = 2 ≠ 8
i= 2   a[i] = 8 = 8
i=     a[i] =
```

so increment i
" "
so success return 2

Find key of 4:

```
i= 0   a[i] = 5 ≠ 4
i= 1   a[i] = 2 ≠ 4
i= 2   a[i] = 8 ≠ 4
i= 3   a[i] = 9 ≠ 4
```

so increment i
"
"
"

no elements left
so return -1

Linear Search: Algorithm

Algorithm: Linear search

Input: list, key

for *each element of the list* **do**
    if *element = key* **then**
        return *index of element*
    end
end

return *-1*

Performance

- To analyse the performance, we can count how many objects are examined by the algorithm
- We are interested in being able to *guarantee* how long the algorithm will take, and in *predicting* how long it will take
- The running time will depend on whether the search is *successful* (the key is found in the list) or *unsuccessful* (the key is not found)

Expected value

Definition (Expected value)

For a given experiment, let X be a random variable with possible values from $\{x_1, \ldots, x_n\}$. The *expected value* of X, denoted $E[X]$, is:

$$E[X] = x_1.P(X = x_1) + x_2.P(X = x_2) + \cdots + x_n.P(X = x_n)$$

Example

Let X be the value rolled on a 6-sided dice:

fair

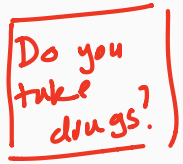
$$E[X] = 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} + \cdots + 6 \cdot \frac{1}{6}$$

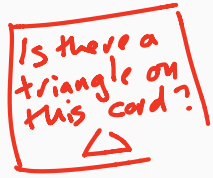
$$= \frac{21}{6} = 3.5$$

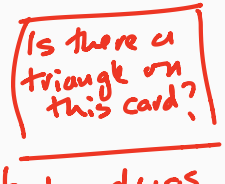
Example: drug testing

From *Why do buses come in threes?*

During the Vietnam war, the US authorities wanted to know how many soldiers used drugs. How could they survey the soldiers without requiring them to admit to a criminal offence?







Let p be the proportion of soldiers that take drugs

$$E[\text{Number of yes from 1 soldier}] = \frac{1}{3} \cdot p \cdot 1 + \frac{1}{3}(1-p) \cdot 0 + \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot 0$$

$$= \frac{p+1}{3}$$

If we ask N soldiers, then

$$E[\text{yesses}] = \frac{N(p+1)}{3}$$

$$50 = \frac{100(p+1)}{3}$$

$$150 = 100p + 100$$

$$p = 0.5$$

Law of Large Numbers

Theorem (Weak law of large numbers)

Let $X_1, \ldots, X_n$ be a sequence of independent and identically distributed random variables, each having mean μ and standard deviation σ. Then:

$$P\left(\left|\frac{X_1 + \cdots + X_n}{n} - \mu\right| < \epsilon\right)$$

how accurately

for any positive ϵ approaches 1 as $n \to \infty$.

alternatively...

If we repeat an experiment[1] enough times, the mean of our sample results will converge to the expected result. I.e. we can measure through experiments instead of finding an analytic solution.

[1] With some assumptions that it is relatively well behaved.

Example: rolling a dice

Linear search performance: successful

Assumptions

Suppose we have a list of n different elements in a random order, and that the key is present.

Suppose the key is equally likely to appear anywhere in the list: $P(key == a[i]) = 1/n$ for $0 \leq i < n$

Let X be the number of elements that need to be compared before finding the key

Average case

$$E[X] = 1 \cdot \frac{1}{n} + 2 \cdot \frac{1}{n} + \cdots + n \cdot \frac{1}{n} = \frac{n(n+1)}{2n} = \frac{n+1}{2}$$

Linear search performance: unsuccessful

On average an unsuccessful search takes n comparisons

In the worst case an unsuccessful search takes n comparisons

– " ——— Successful – " ——— n "

Linear search for sorted lists

Suppose the list we wish to search is in ascending order. We could stop our search as soon as we encounter an element larger than the key:

Algorithm: Linear search (sorted)

Input: sorted (ascending) list, key

```
for each element of the list do
    if element = key then
        return index of element
    else if element > key then
        return -1
end
return -1
```

Note: if element >= key:
if ...
...

Performance

How does this change affect the *worst case* and *average* performance?

Average successful stays the same

" unsuccessful has gone from $n \rightarrow n/2$

Worst case stays the same

Binary search

- *Binary search* applies the principle of *divide and conquer* to give a more efficient search algorithm
- At each step the algorithm eliminates half of the elements from the search
- In order to do this the list is **required** to be sorted

Example: successful

Example

List: 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151
Key : 101

Pick middle 82 < key so search {101, 112, 125, 131, 142, 151}
Pick " 125 > key so search {101, 112}
" " 101 == key so return position

Example: unsuccessful

Example

```
List: 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151
Key  : 21
```

Pick middle 82 > key so search {0, 7, 9, 23, 54}

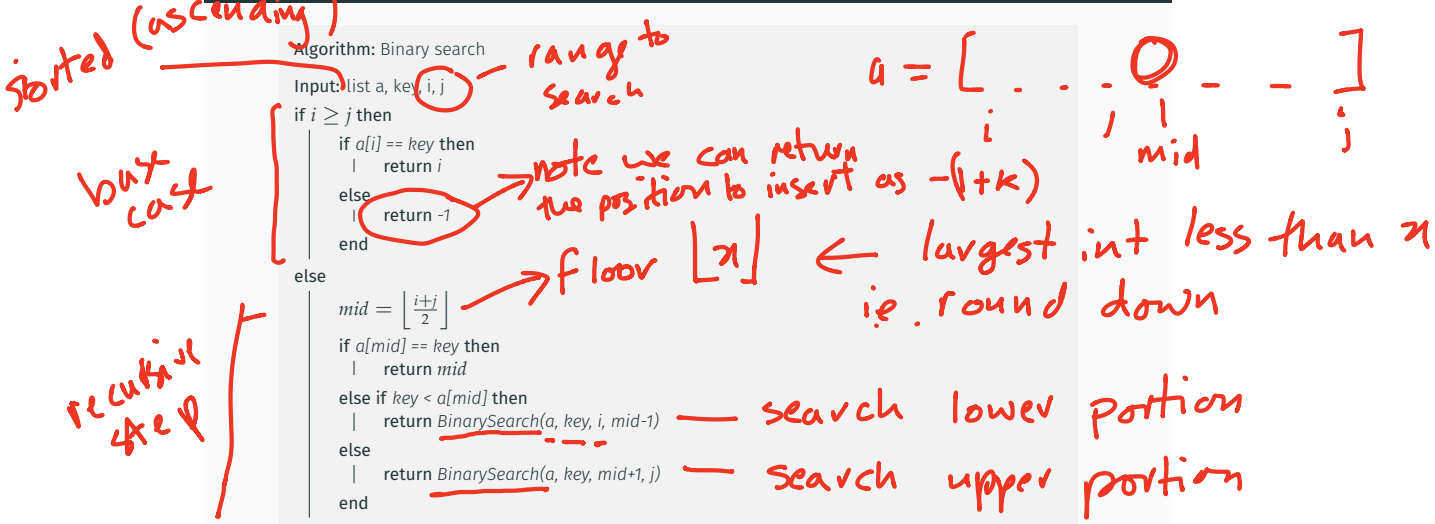
" " 9 < key so search {23, ~~54~~}

" " 23 > key so search {}

No where left to search so return -1

Binary search algorithm

```
Algorithm: Binary search
Input: list a, key, i, j
if i ≥ j then
    if a[i] == key then
        return i
    else
        return -1
    end
else
    mid = ⌊(i+j)/2⌋
    if a[mid] == key then
        return mid
    else if key < a[mid] then
        return BinarySearch(a, key, i, mid-1)
    else
        return BinarySearch(a, key, mid+1, j)
    end
end
```



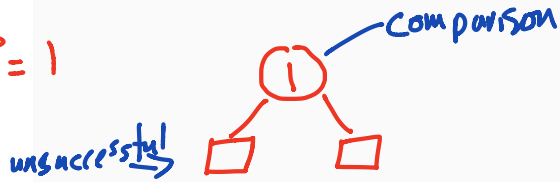
Performance of binary search

Consider a list a of $n = 2^{k-1}$ elements for $k = 1, 2, 3, \ldots$

Suppose $a = [1, \ldots, n]$

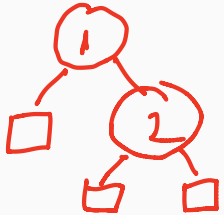
$k = 1$ $n = 2^0 = 1$

$a = [1]$



$k = 2$ $n = 2^1 = 2$

$a = [1, 2]$



Maximum comparisons

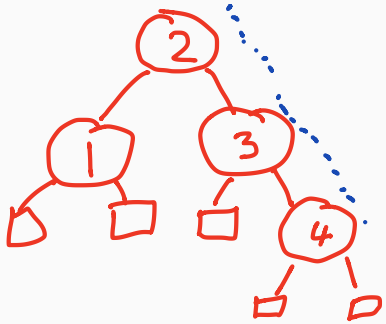
1

2

Performance of binary search

$k=3 \quad n=2^2=4$

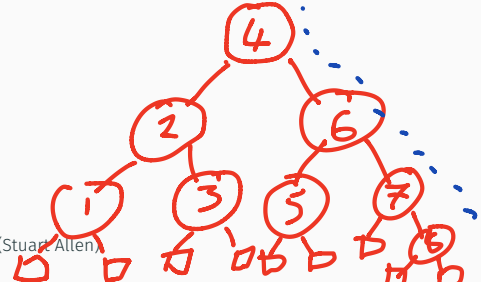
$a=[1,2,3,4]$



3

$k=4 \quad n=2^3=8$

$a=[1,\ldots,8]$



4

Summary

- Linear search algorithm
- Binary search algorithm
- How should we compare the performance of algorithms
- Analysis of the performance of linear vs binary search