

## Searching and Sorting

## Outline

1 Searching

2 Sorting



## Searching

The search problem involves searching for a key in a collection of  $n$  keys

## Searching

The search problem involves searching for a key in a collection of  $n$  keys

### Linear search

linearsearch.py

```
from instream import InStream
import stdio
import sys

def indexOf(a, key):
    for i in range(len(a)):
        if a[i] == key:
            return i
    return -1

def _main():
    inStream = InStream(sys.argv[1])
    whiteList = inStream.readAllInts()
    while not stdio.isEmpty():
        key = stdio.readInt()
        if indexOf(whiteList, key) == -1:
            stdio.writeln(key)

if __name__ == '__main__':
    _main()
```

## Searching

The search problem involves searching for a key in a collection of  $n$  keys

### Linear search

linearsearch.py

```
from instream import InStream
import stdio
import sys

def indexOf(a, key):
    for i in range(len(a)):
        if a[i] == key:
            return i
    return -1

def _main():
    inStream = InStream(sys.argv[1])
    whiteList = inStream.readAllInts()
    while not stdio.isEmpty():
        key = stdio.readInt()
        if indexOf(whiteList, key) == -1:
            stdio.writeln(key)

if __name__ == '__main__':
    _main()
```

Running time:  $T(n) = n$  (linear)



## Searching

```
>_ ~/workspace/ipp/programs
```

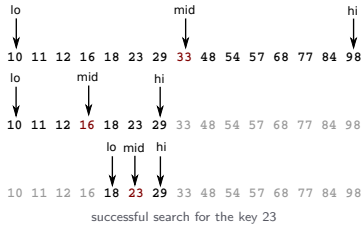
```
$ /usr/bin/time --format='%e seconds' python3 linearsearch.py ../data/tinyW.txt < ../data/tinyT.txt
50
99
13
0.05 seconds
$ /usr/bin/time --format='%e seconds' python3 linearsearch.py ../data/largeW.txt < ../data/largeT.txt
Takes way too long
```





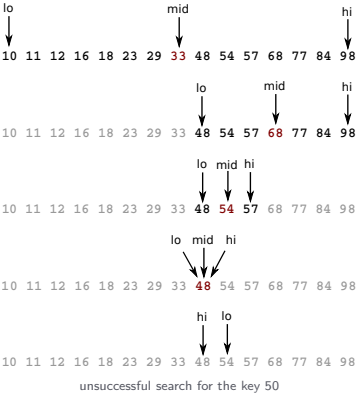
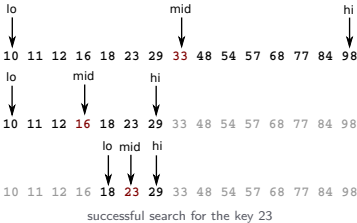
# Searching

## Binary search



# Searching

## Binary search





## Searching

binarysearch.py

```
from instream import InStream
import stdio
import sys

def indexOf(a, key):
    lo = 0
    hi = len(a) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key < a[mid]:
            hi = mid - 1
        elif key > a[mid]:
            lo = mid + 1
        else:
            return mid
    return -1

def _main():
    inStream = InStream(sys.argv[1])
    whiteList = inStream.readAllInts()
    whiteList.sort()
    while not stdio.isEmpty():
        key = stdio.readInt()
        if indexOf(whiteList, key) == -1:
            stdio.writeln(key)

if __name__ == '__main__':
    _main()
```



## Searching

```
>_ ~/workspace/ipp/programs
```

```
$ /usr/bin/time --format='%e seconds' python3 binarysearch.py ../data/tinyW.txt < ../data/tinyT.txt
50
99
13
0.05 seconds
$ /usr/bin/time --format='%e seconds' python3 binarysearch.py ../data/largeW.txt < ../data/largeT.txt
...
29798919
9505145
32449528
38862597
69830567
75.47 seconds
```

## Searching

```
>_ ~/workspace/ipp/programs
```

```
$ /usr/bin/time --format='%e seconds' python3 binarysearch.py ../data/tinyW.txt < ../data/tinyT.txt  
50  
99  
13  
0.05 seconds  
$ /usr/bin/time --format='%e seconds' python3 binarysearch.py ../data/largeW.txt < ../data/largeT.txt  
...  
29798919  
9505145  
32449528  
38862597  
69830567  
75.47 seconds
```

Running time:  $T(n) = \log n$  (logarithmic)



## Sorting

## Sorting

The sort problem involves rearranging a sequence of objects so as to put them in some logical order

## Sorting

The sort problem involves rearranging a sequence of objects so as to put them in some logical order

Insertion sort is similar to sorting a bridge hand — consider the cards one at a time, inserting each into its proper place among those already considered

## Sorting

The sort problem involves rearranging a sequence of objects so as to put them in some logical order

Insertion sort is similar to sorting a bridge hand — consider the cards one at a time, inserting each into its proper place among those already considered

Running time:  $T(n) = n^2$  (quadratic)

## Sorting

## Sorting

insertion.py

```
import stdio
import sys

def sort(a, key=None):
    n = len(a)
    for i in range(1, n):
        for j in range(i, 0, -1):
            v, w = a[j], a[j - 1]
            if key:
                v, w = key(v), key(w)
            if v >= w:
                break
            _exchange(a, j, j - 1)

def _exchange(a, i, j):
    temp = a[i]
    a[i] = a[j]
    a[j] = temp

def _main():
    a = stdio.readAllStrings()
    if sys.argv[1] == '--':
        sort(a, key=lambda x: x.lower())
    elif sys.argv[1] == '+':
        sort(a)
    else:
        raise Exception('Illegal command-line argument')
    for s in a:
        stdio.write(s + ' ')
    stdio.writeln()

if __name__ == '__main__':
    _main()
```

## Sorting

## Sorting

```
>_ ~/workspace/ipp/programs
```

```
$ python3 insertion.py -  
I n s e r t i o n S o r t  
<ctrl-d>  
e I i n n o o r r s S t t  
$ python3 insertion.py +  
I n s e r t i o n S o r t  
<ctrl-d>  
I S e i n n o o r r s t t
```



## Sorting

## Sorting

Trace

		a[]											
i	j	0	1	2	3	4	5	6	7	8	9	10	
		S	O	R	T	E	X	A	M	P	L	E	
1	0	O	S	R	T	E	X	A	M	P	L	E	
2	1	O	R	S	T	E	X	A	M	P	L	E	
3	3	O	R	S	T	E	X	A	M	P	L	E	
4	0	E	O	R	S	T	X	A	M	P	L	E	
5	5	E	O	R	S	T	X	A	M	P	L	E	
6	0	A	E	O	R	S	T	X	M	P	L	E	
7	2	A	E	M	O	R	S	T	X	P	L	E	
8	4	A	E	M	O	P	R	S	T	X	L	E	
9	2	A	E	L	M	O	P	R	S	T	X	E	
10	2	A	E	E	L	M	O	P	R	S	T	X	
		A	E	E	L	M	O	P	R	S	T	X	

## Sorting

## Sorting

Merge sort is based on a simple operation known as merging: combining two ordered lists to make one larger ordered list

## Sorting

Merge sort is based on a simple operation known as merging: combining two ordered lists to make one larger ordered list

To sort a list, we divide it into two halves, sort the two halves recursively, and then merge the results

input	M	E	R	G	E	S	O	R		T	E	X	A	M	P	L	E
sort left half	E	E	G	M	O	R	R	S		T	E	X	A	M	P	L	E
sort right half	E	E	G	M	O	R	R	S		A	E	E	L	M	P	T	X
merge results	A	E	E	E	E	G	L	M		M	O	P	R	R	S	T	X

## Sorting

Merge sort is based on a simple operation known as merging: combining two ordered lists to make one larger ordered list

To sort a list, we divide it into two halves, sort the two halves recursively, and then merge the results

input	M	E	R	G	E	S	O	R		T	E	X	A	M	P	L	E
sort left half	E	E	G	M	O	R	R	S		T	E	X	A	M	P	L	E
sort right half	E	E	G	M	O	R	R	S		A	E	E	L	M	P	T	X
merge results	A	E	E	E	E	G	L	M		M	O	P	R	R	S	T	X

Running time:  $T(n) = n \log n$  (linearithmic)

## Sorting

## Sorting

merge.py

```
import ndarray
import stdio
import sys

def sort(a, key=None):
    aux = ndarray.create1D(len(a), None)
    _sort(a, aux, 0, len(a) - 1, key)

def _sort(a, aux, lo, hi, key=None):
    if hi <= lo:
        return
    mid = lo + (hi - lo) // 2
    _sort(a, aux, lo, mid, key)
    _sort(a, aux, mid + 1, hi, key)
    _merge(a, aux, lo, mid, hi, key)

def _merge(a, aux, lo, mid, hi, key=None):
    for k in range(lo, hi + 1):
        aux[k] = a[k]
    i, j = lo, mid + 1
    for k in range(lo, hi + 1):
        if i > mid:
            a[k] = aux[j]
            j += 1
        elif j > hi:
            a[k] = aux[i]
            i += 1
        else:
            v, w = aux[i], aux[j]
            if key:
                v, w = key(v), key(w)
            if w < v:
                a[k] = aux[j]
                j += 1
            else:
```



## Sorting

merge.py

```
        a[k] = aux[i]
        i += 1

def _main():
    a = stdio.readAllStrings()
    if sys.argv[1] == '-':
        sort(a, key=lambda x: x.lower())
    elif sys.argv[1] == '+':
        sort(a)
    else:
        raise Exception('Illegal command-line argument')
    for s in a:
        stdio.write(s + ' ')
    stdio.writeln()

if __name__ == '__main__':
    _main()
```

## Sorting

## Sorting

```
>_ ~/workspace/ipp/programs
```

```
$ python3 merge.py -  
M e r g e S o r t  
M e e g r S o r t  
$ python3 merge.py +  
M e r g e S o r t  
M S e e g o r r t
```

## Sorting

## Sorting

Trace (`_sort()`)

	a[]															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
_merge(a, 0, 0, 1, aux)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
_merge(a, 2, 2, 3, aux)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
_merge(a, 0, 1, 3, aux)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
_merge(a, 4, 4, 5, aux)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
_merge(a, 6, 6, 7, aux)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
_merge(a, 4, 5, 7, aux)	E	G	M	R	E	O	R	S	T	E	X	A	M	P	L	E
_merge(a, 0, 3, 7, aux)	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
_merge(a, 8, 8, 9, aux)	E	E	G	M	O	R	R	S	E	T	X	A	M	P	L	E
_merge(a, 10, 10, 11, aux)	E	E	G	M	O	R	R	S	E	T	A	X	M	P	L	E
_merge(a, 8, 9, 11, aux)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
_merge(a, 12, 12, 13, aux)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
_merge(a, 14, 14, 15, aux)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	E	L
_merge(a, 12, 13, 15, aux)	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
_merge(a, 8, 11, 15, aux)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
_merge(a, 0, 7, 15, aux)	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X