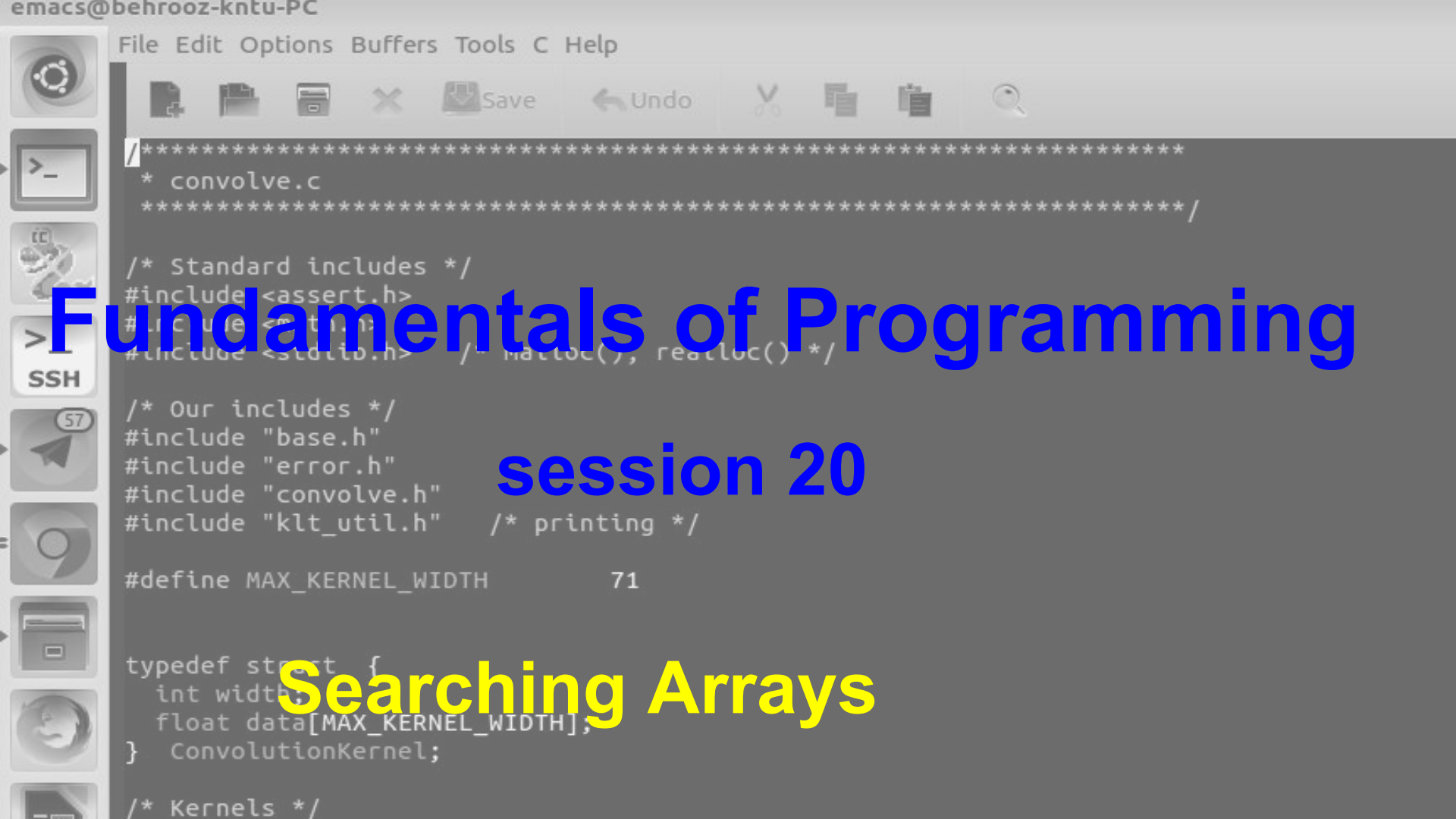

emacs@behrooz-kntu-PC
File Edit Options Buffers Tools C Help
Save
Undo
/* Standard includes */
#include <assert.h>
/* Our includes */
#include "base.h"
#include "error.h"
#include "convolve.h"
#include "klt_util.h" /* printing */
#define MAX_KERNEL_WIDTH 71
typedef struct {
int width;
float data[MAX_KERNEL_WIDTH];
} ConvolutionKernel;
/* Kernels */
Fundamentals of Programming
session 20
Searching Arrays

Mean, median and mode

Write a program printing mean, median, and mode of an arrays of integers. You can assume that array elements are integers between 0 and 19.

stats.c

```
void printArray(const int a[], int n);
void sort(int a[], int n);
double mean(const int a[], int n);
double median(int a[], int n);
int mode(const int a[], int n);

int main() {
  int a[] = {1,18,2,3,4,5,7,7,4,19,18,12,13, 0, 18,
             14,17,1,0,8,9,9,9,10, 11, 0,19,18,
             18,1,3,4,6,7,8,3,2,15,1,0,7,13,14,
             10,12,16,17,18,1,4,6,8,4,0,5,8,7,5,
             6,4,9,16,15,12,13,14,12,13,16,18,9,10};

  int n = sizeof(a) / sizeof(a[0]);

  printArray(a,n);

  printf("mean=%.2f, mode= %d, median=%.1f\n", mean(a,n), mode(a,n), median(a,n));

  return 0;
}
```

Mean, median and mode

Write a program printing mean, median, and mode of an arrays of integers. You can assume that array elements are integers between 0 and 19.

```c
double mean(const int a[], int n) {

  int sum = 0;
  for (int i = 0; i < n; i++)
    sum += a[i];

  return sum / (double) n;
}
```

stats.c

Mean, median and mode

Assume that the array elements are integers between **0 and 19**.

```
int mode(const int a[], int n) {
  int count[20];

  for (int i = 0; i < 20; i++)
    count[i] = 0;

  for (int i = 0; i < n; i++)
    count[a[i]]++;

  int maxi = 0;
  for (int i = 1; i < 20; i++)
    if (count[i] > count[maxi])
      maxi = i;

  return maxi;
}
```

stats.c

Static vs automatic arrays

```c
int mode(const int a[], int n) {
  static int count[20];

  for (int i = 0; i < 20; i++)
    count[i] = 0;

  for (int i = 0; i < n; i++)
    count[a[i]]++;

  int maxi = 0;
  for (int i = 1; i < 20; i++)
    if (count[i] > count[maxi])
      maxi = i;

  return maxi;
}
```

stats2.c

Median

```c
double median(int a[], int n) {
  sort(a,n);

  if (n % 2 == 0)
    return (a[n/2] + a[n/2-1]) / 2.0;
  else
    return a[n/2];

}
```

stats.c

Median

```c
double median(int a[], int n) {
  sort(a,n);

  if (n % 2 == 0)
    return (a[n/2] + a[n/2-1]) / 2.0;
  else
    return a[n/2];

}
```

stats.c

side effect?

Median

```c
double median(const int a[], int n) {
  int b[n];

  for (int i = 0; i < n; i++)
    b[i] = a[i];

  sort(b,n);

  if (n % 2 == 0)
    return (b[n/2] + b[n/2-1]) / 2.0;
  else
    return b[n/2];

}
```

stats2.c

Searching arrays

25	19	14	18	27	6	32	18	20	1	21

- Is the number 19 in the array?
- What is its index?

```
index = search(key, array, size);
```

linear search

```c
int linearsearch(int key, const int array[], int size) {
  for (int i = 0; i < size; i++)
    if (array[i] == key)
      return i;
  return -1;
}
```

linearsearch.c

linear search

```c
void printArray(const int array[], int n);
int linearsearch(int key, const int array[], int size);

int main() {
  int array[] = {25, 19, 14, 18, 27, 6, 32, 18, 20, 1, 21};
  int size = sizeof(array) / sizeof(array[0]);
  int key;

  printArray(array,size);

  while (1) {
    scanf("%d", &key);

    if (key == -1)
      break;

    int index = linearsearch(key, array, size);

    printf("%d\n", index);
  }

  return 0;
}


int linearsearch(int key, const int array[], int size) {
  for (int i = 0; i < size; i++)
    if (array[i] == key)
      return i;
  return -1;
}
```

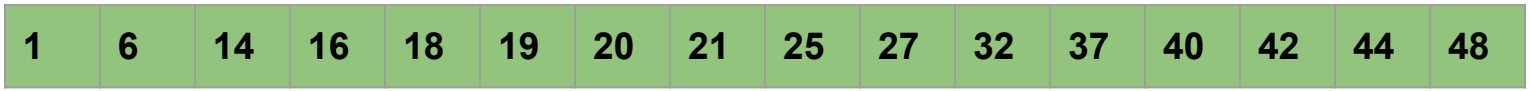
linearsearch.c

linear search

- Average no. of elements to examine
- What if the array is sorted?

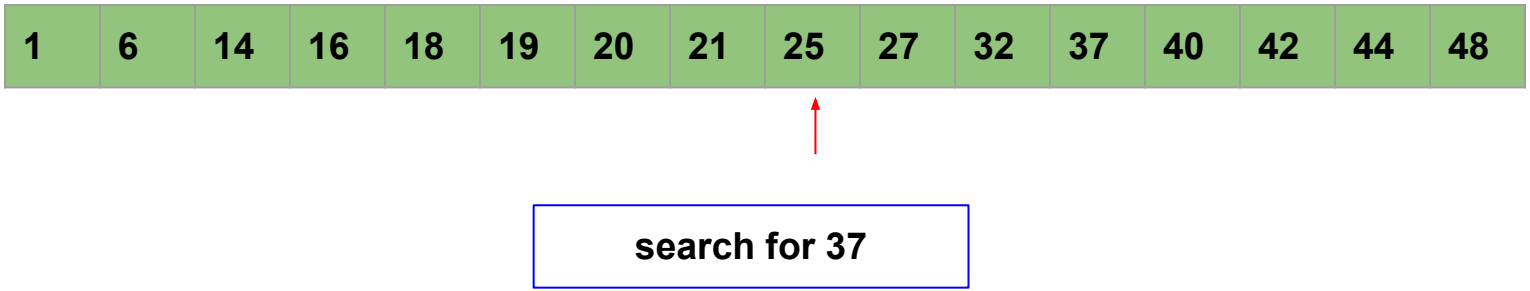
1	6	14	16	18	19	20	21	25	27	32

Binary search

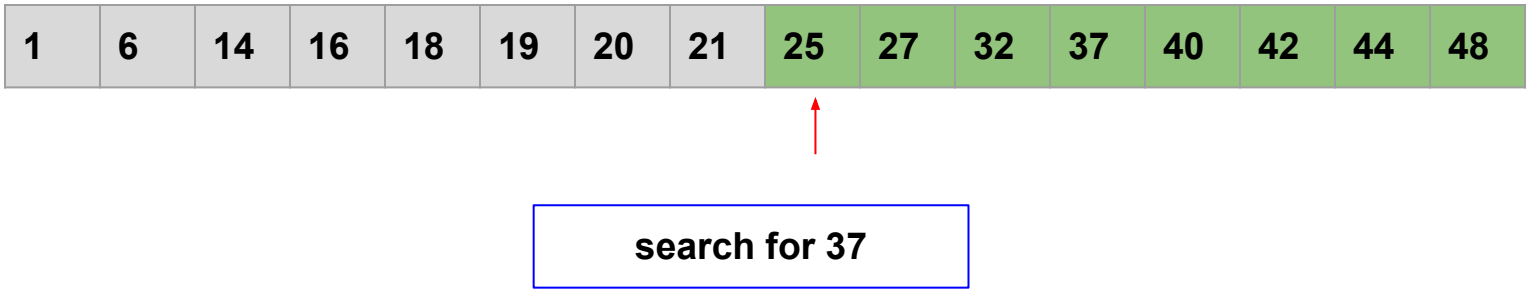


search for 37

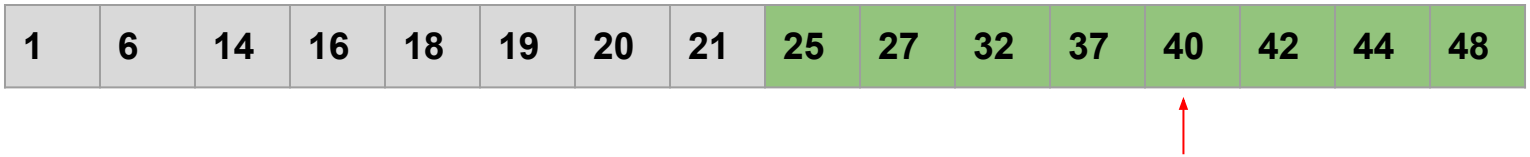
Binary search



Binary search

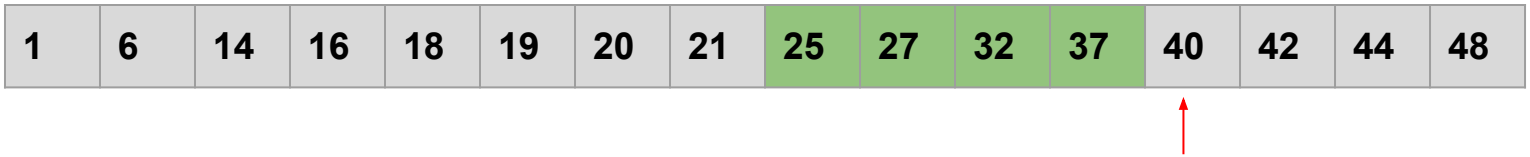


Binary search



search for 37

Binary search



search for 37

Binary search

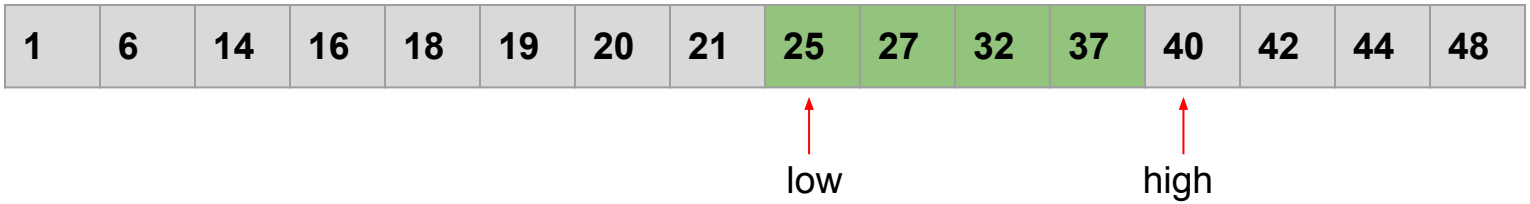
Binary search

Binary search

Binary search

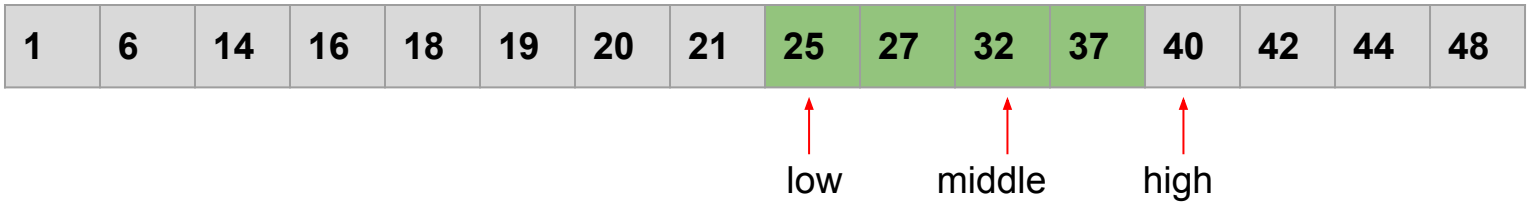
Binary search

```
int binarysearch(int key, const int array[], int size) {
  int low = 0;
  int high = size;

  // binary search algorithm


}
```



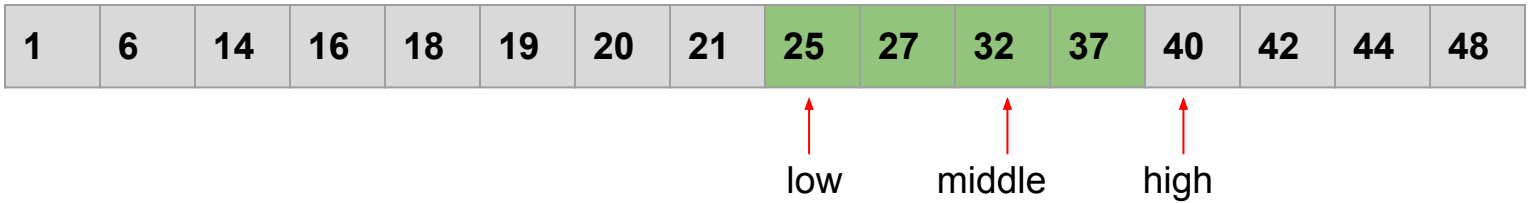
Binary search

```
int binarysearch(int key, const int array[], int size) {
  int low = 0;
  int high = size;

  while (???) {

    int middle = (high + low)/2;

    if (key < array[middle]) {

    }
    else {

    }

  }

}
```



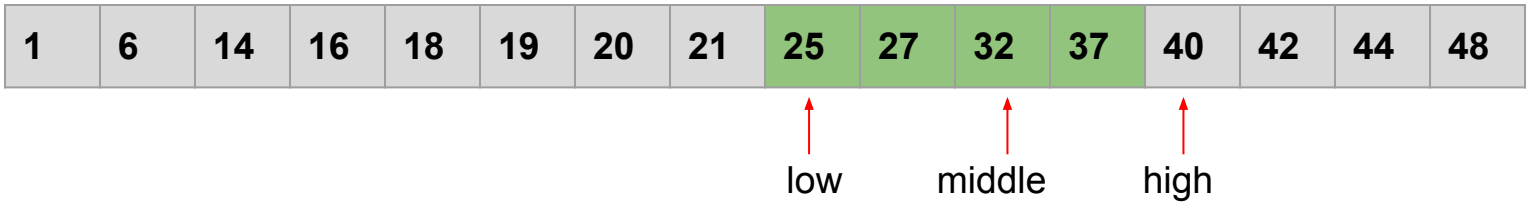
Binary search

```
int binarysearch(int key, const int array[], int size) {
  int low = 0;
  int high = size;

  while (???) {

    int middle = (high + low)/2;

    if (key < array[middle]) {
      high = middle;
    }
    else {
      low = middle;
    }

  }

}
```



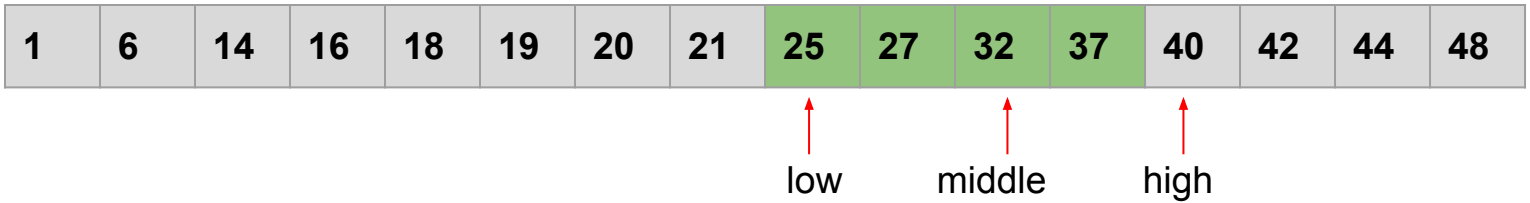
Binary search

```
int binarysearch(int key, const int array[], int size) {
  int low = 0;
  int high = size;

  while (high > low+1) {

    int middle = (high + low)/2;

    if (key < array[middle]) {
      high = middle;
    }
    else {
      low = middle;
    }

  }

}
```



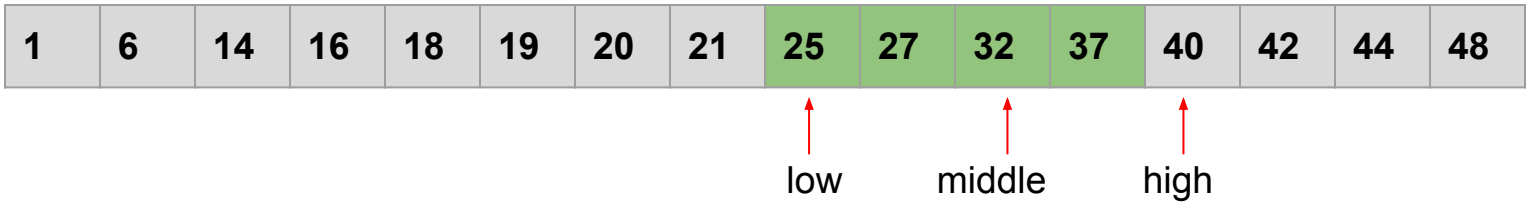
Binary search

```
int binarysearch(int key, const int array[], int size) {
  int low = 0;
  int high = size;

  while (high > low+1) {

    int middle = (high + low)/2;

    if (key < array[middle]) {
      high = middle;
    }
    else {
      low = middle;
    }

  }

  if (array[low] == key)
    return low;
  else
    return -1;

}
```



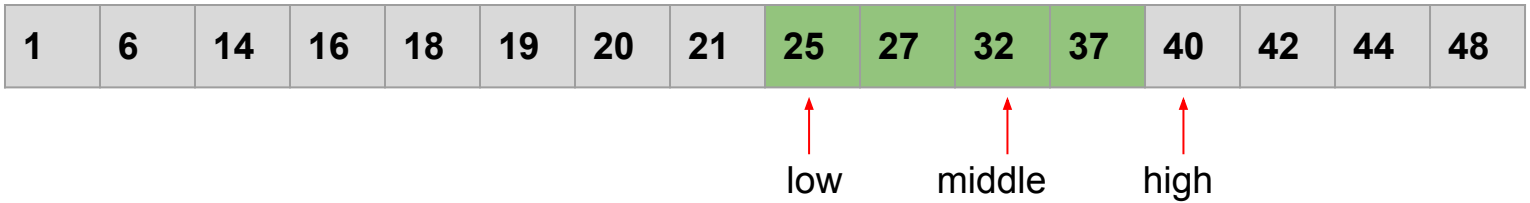
Binary search

```c
int binarysearch(int key, const int array[], int size) {
  int low = 0;
  int high = size;

  while (high > low+1) {

    int middle = (high + low)/2;

    if (key < array[middle]) {
      high = middle;
    }
    else {
      low = middle;
    }

  }

  return (array[low] == key) ? low : -1;

}
```



```c
int binarysearch(int key, const int array[], int size) {
  int low = 0;
  int high = size;

  while (high > low+1) {

    int middle = (high + low)/2;

    if (key < array[middle]) {
      high = middle;
    }
    else {
      low = middle;
    }

  }

  return (array[low] == key) ? low : -1;

}
```

- Assume size > 0
  - Can we ever have **high <= low**?
  - Why high = low + 1 out of the loop?
  - What if size == 1?
- What if size == 0?
- How many comparisons?
- Compare to linear search
  - size = 64
  - size = 1024
  - size ≈ 1000,000,000
- Sorting the array first?



2D arrays

0	0	0	0	0	0
0	1	2	3	4	5
0	2	4	6	8	10
0	3	6	9	12	15
0	4	8	12	16	20
0	5	10	15	20	25

2D arrays

0	0	0	0	0	0
0	1	2	3	4	5
0	2	4	6	8	10
0	3	6	9	12	15
0	4	8	12	16	20
0	5	10	15	20	25

$$A = \begin{pmatrix} 3 & -5 & 4 \\ 9 & 8 & -7 \\ -6 & 4 & 2 \end{pmatrix}, B = \begin{pmatrix} -2 & -1 & 1 \\ 5 & -7 & 6 \\ 9 & 3 & 2 \end{pmatrix}$$

https://advancedmathclubsk.weebly.com/matrices.html

2D arrays

https://hiveminer.com/Tags/desert,isfahan/Recent