Algorithms

Date	@July 6, 2023
# R	8
# R (2)	
# R (3)	
# R(1)	
R1	
R2	
R3	

▼ Main points

- Introduction
- Searching algorithms
  - Linear search
  - Binary search
- Sorting algorithms
  - Selection sort
  - Bubble sort

▼ Algorithms introduction

▼ History of algorithms

-

▼ Algorithms introduction 1

▼ code

```
int func(int);
int main()
{
  func(5);
```

```
  return 0;
}

int func(int x)
{
  int y;
  y = x + 10;
  return y;
}
```

▼ output

```
15
```

▼ notes

- Take the same time every time
- constant execution time
- big o notation = 1

▼ Algorithms introduction 2

▼ code

```
void func(int*,int);

int main()
{
    int arr[8] = {1,2,0,4,5,5,6,7};
    func(&arr,8);
    return 0;
}
    void func(int *p,int size)
    {
    int i;
    for(i=0;i<size;i++){
    if(p[i]==0)
    return;}
    }
```

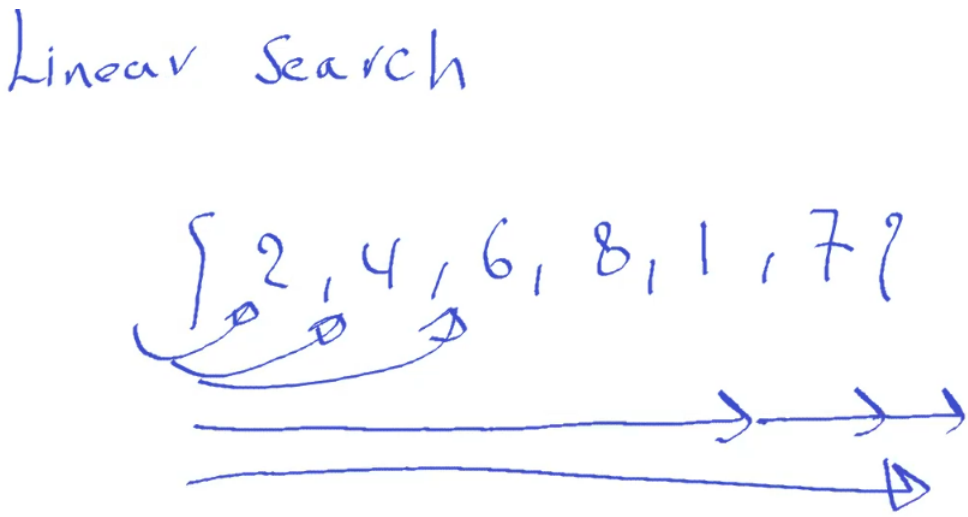
- output

```
no output
```

- notes
  - Take different time every time
  - Depending on the position of the 0 in the array
  - Worst case scenario = 0 is the last element in array or no 0
  - big o notation = o(n) where n is the number of array elements

Searching algorithms

- Explain how linear search work



- What is big O of linear search
  - O(n)
- Linear Search
  - code

```
void linearSearch(int *arr,int size, int target);

int main()
{
    int arr[5] = {1,2,3,4,5};
```

```
    linearSearch(arr, 5,5);

}

void linearSearch(int *arr,int size, int target)
{
    int i;
    for(i = 0; i < size; i++)
    {
        if(arr[i] == target)
        {
            printf("The target number %d is found in position %d \n",arr[i],
            return;
        }
    }
    printf("The number is not found");
}
```

- output

```
The target number 5 is found in position 4
```

- Which is faster linear or binary search?
  - binary search
- What is the problem of binary search?
  - You must sort the data first
- Explain how binary search work?

- What is big O of binary search?
  - Log(n)
- Binary Search
  - code

```
void binarySearch(int *,int, int);

int main()
{
    int arr[5] = {1,2,3,4,5};
    binarySearch(arr, 5,5);
```

```
}

void binarySearch(int *arr,int size, int target)
{
    int first,last,middle;
    first  = 0;
    last = size  - 1;
    //This is because if last > first this meant hat the data is not found
    while(first <= last)
    {
        middle = (first + last) / 2;
        if(target == arr[middle])
        {
            printf("The target %d is found in position %d \n",arr[middle],mi
            return;
        }
        else if(target > arr[middle])
        {
            first = middle + 1;
        }
        else
        {
            last = middle - 1;
        }
    }
    printf("The number is not found");
}
```

▼ output

```
The target number 5 is found in position 4
```

▼ Sorting algorithms

▼ How does selection sorting algorithm work?

```
4       7       3       5       2
3       7       4       5       2
3       7       4       5       2
2       7       4       5       3


2       4       7       5       3
2       4       7       5       3
2       3       7       5       4


2       3       5       7       4
2       3       4       7       5
```

```
2       3       4       5       7
```

- btdwr 3la as8r wa7ed kol mara w tgebo n7yt el4mal

▼ How the bubble sorting algorithm work?

- It works on the opposite way
- Dwr 3la akbr rkm w t7to n7yt elymen w n2s elarray 5ana

▼ Selection sort

▼ code

```
void selectionSort(int *, int);

int main()
{
    int arr[5] = {7,4,3,5,2};
    int i;
    printf("The array after sorting : \n");
    selectionSort(arr, 5);
    for(i = 0; i < 5; i++)
    {
        printf("%d\t",arr[i]);
    }


}
void selectionSort(int * arr, int size)
{
    int i,j,temp;
    for(i = 0; i < size - 1; i++)
        for(j = i+1; j < size; j++) //j = 1 is wrong
        {
            if(arr[i] > arr[j])
            {
               temp = arr[i];
               arr[i] = arr[j];
               arr[j] = temp;
            }

        }
}
```

▼ output

```
The array after sorting :
2       3       4       5       7
```

▼ Bubble sort

▼ code

```
void bubbleSort(int *, int);

int main()
{
    int arr[5] = {7,4,3,5,2};
    int i;
    printf("The array after sorting : \n");
    bubbleSort(arr, 5);
    for(i = 0; i < 5; i++)
    {
        printf("%d\t",arr[i]);
    }


}
void bubbleSort(int * arr, int size)
{
    int i,j,temp;
    for(i = 0; i < size - 1; i++)
        for(j = 0; j < size-1-i; j++)
        {
            if(arr[j] > arr[j+1])
            {
               temp = arr[j];
               arr[j] = arr[j+1];
               arr[j+1] = temp;
            }

        }
}
```

▼ output

```
The array after sorting :
2       3       4       5       7
```

▼ notes

- size - 1 ⇒ lase one will be already sorted
- size - 1 - i ⇒ array first time decreases by zero then 1 , ….. which is equal to i

▼ What is the big O of selection and bubble sort algorithms ?

- n^2 where n is the number of array elements

▼ Exercises

▼ 1

▼ code

```
# include <stdio.h>
# define _ " Hi "
int main()
{
    printf( );
    return 0;
}
```

▼ output

```
Hi
```

▼ 2

▼ code

```
#define max abc
#define abc 100

int main()
{
  printf("maximum is %d", max);
}
```

▼ output

```
100
```

- 3
  - code

```
#define char int

int main()
{
  char x;
  printf("%d byte",sizeof(x));
}
```

  - output

```
4  byte
```

- 4
  - code

```
#define square(x) x*x
int main()
{
  int x;
  x = 36/square(6);
  printf("%d", x);
}
```

  - output

```
36 //36 / 6 * 6
//notice the precednece
```

  - notes
    - This fault in thinkin g is due to underestimating the question
    - This informs the importance of analyzing each line of code and thinking of all possible of faults

▼ 5

▼ code

```
#define MAIN int main(){
#define x 10
MAIN printf("%d", x); return 0;}
```

▼ output

```
10
```

▼ 6

▼ code

```
#define SQR(x)   (x*x)
int main()
{
  int a;
  int b = 4;
  a=SQR(b + 2);
  printf("%d\n", a);
}
```

▼ output

```
14
//4 + 2*4 + 2
```

▼ 7

▼ code

```
#define MAX 1000
int main()
{
  int MAX = 100;
```

```
  printf("%d", MAX);
  return 0;
}
```

▼ output

```
error
int 1000 = 100; ???
```

▼ Notes

- Data structure (stack-queue-linked list) is forbidden
- Dynamic memory allocation is forbidden