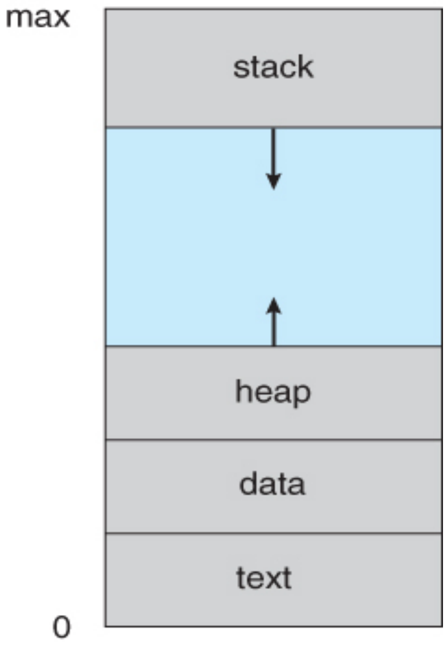
A Process in Memory

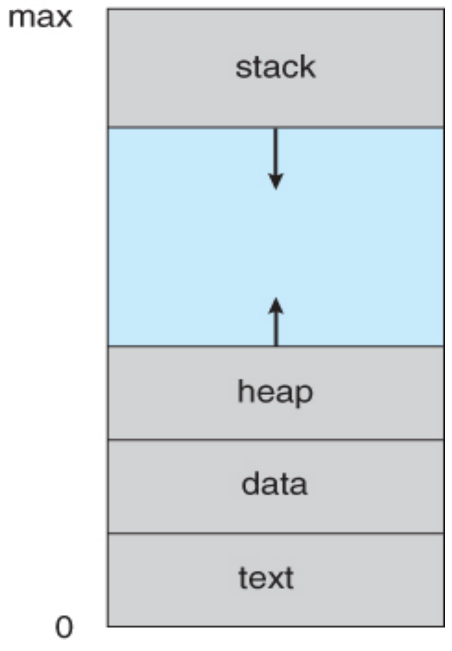
For understanding space complexity, we should first understand how a process (a program in execution) is defined in memory.



A Process in Memory

For understanding space complexity, we should first understand how a process (a program in execution) is defined in memory.

When a process is created by the operating system, a chunk of memory is allocated to the process.



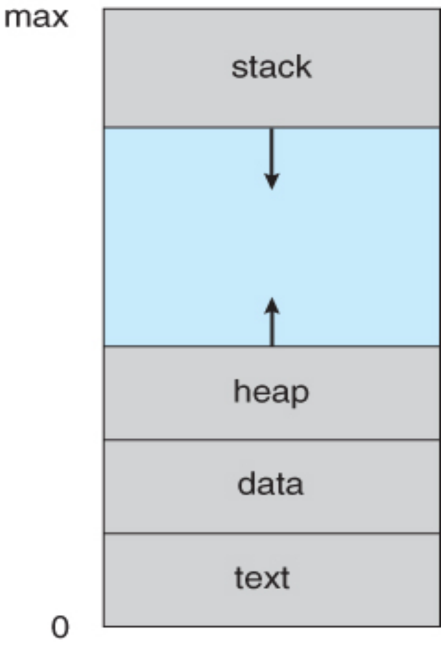
A Process in Memory

For understanding space complexity, we should first understand how a process (a program in execution) is defined in memory.

When a process is created by the operating system, a chunk of memory is allocated to the process.

This chunk is broadly divided into four sections – *text/code, data, heap,* and *stack*.

- **The text/code section stores the executable code of the program that the process will run.**
- **The data section stores the global variables defined in the program.**
- **The heap will store all the dynamically allocated memory during the execution of the program.**
- **The stack section stores Activation records of the active functional calls.**



A Process in Memory

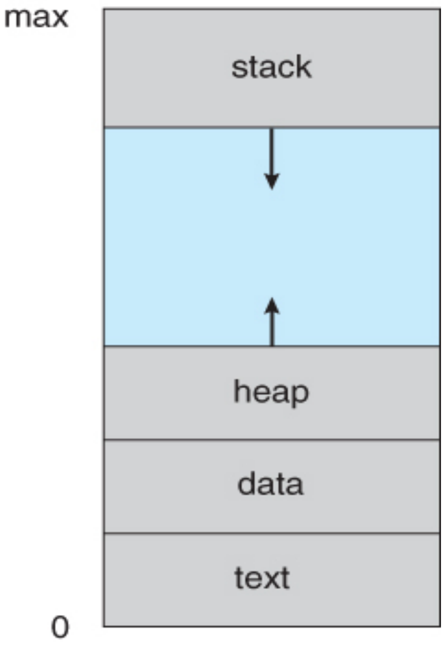
For understanding space complexity, we should first understand how a process (a program in execution) is defined in memory.

When a process is created by the operating system, a chunk of memory is allocated to the process.

This chunk is broadly divided into four sections – *text/code, data, heap,* and *stack*.

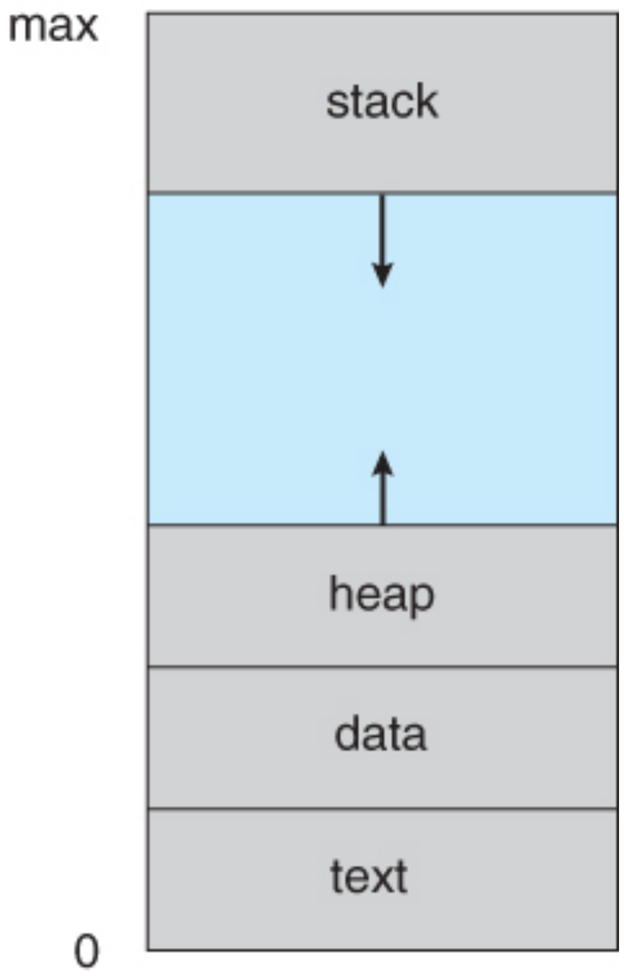
- **The text/code section stores the executable code of the program that the process will run.**
- **The data section stores the global variables defined in the program.**
- **The heap will store all the dynamically allocated memory during the execution of the program.**
- **The stack section stores Activation records the active functional calls.**

Activation record **of a function contains all the local variables defined within the scope of the function, return address to the caller function, etc.**



A Process in Memory

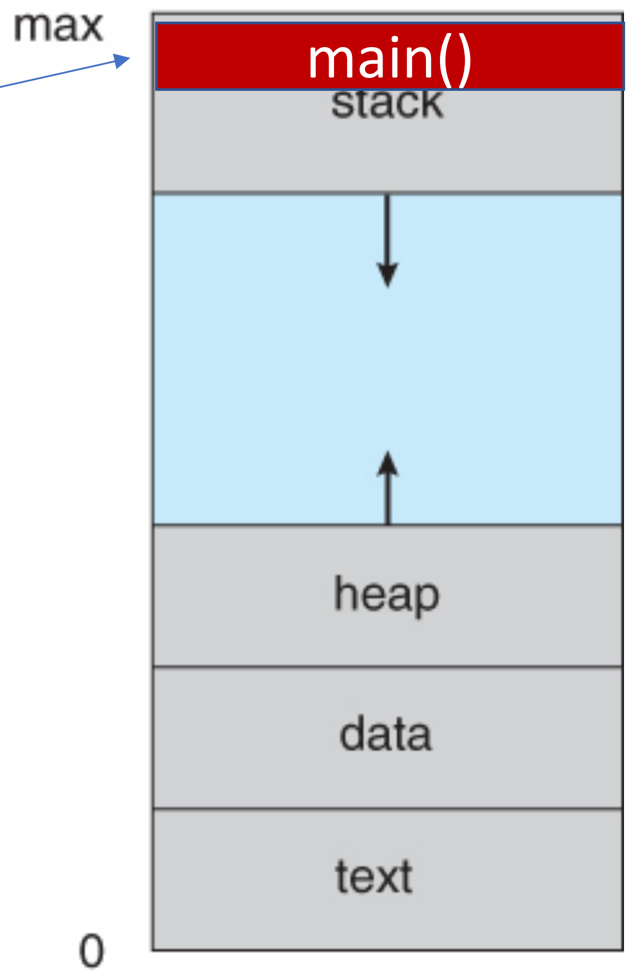
```
#include <stdio.h>
int factorial(int);
int n = 6;
main(){
        int fac;
      fac = factorial(n);
        printf("The  factorial  is  %d",
   fac);
}

int factorial(int n){
        int fac = 1;
        while (n >= 1) {
                  fac = fac * n;
                  n = n-1;
```



A Process in Memory

```
#include <stdio.h>
int factorial(int);
int n=6;
main(){
        int fac;
      fac = factorial(n);
        printf("The factorial is %d",
   fac);
}

int factorial(int n){
        int fac = 1;
        while (n >= 1) {
                  fac = fac * n;
                  n = n-1;
```

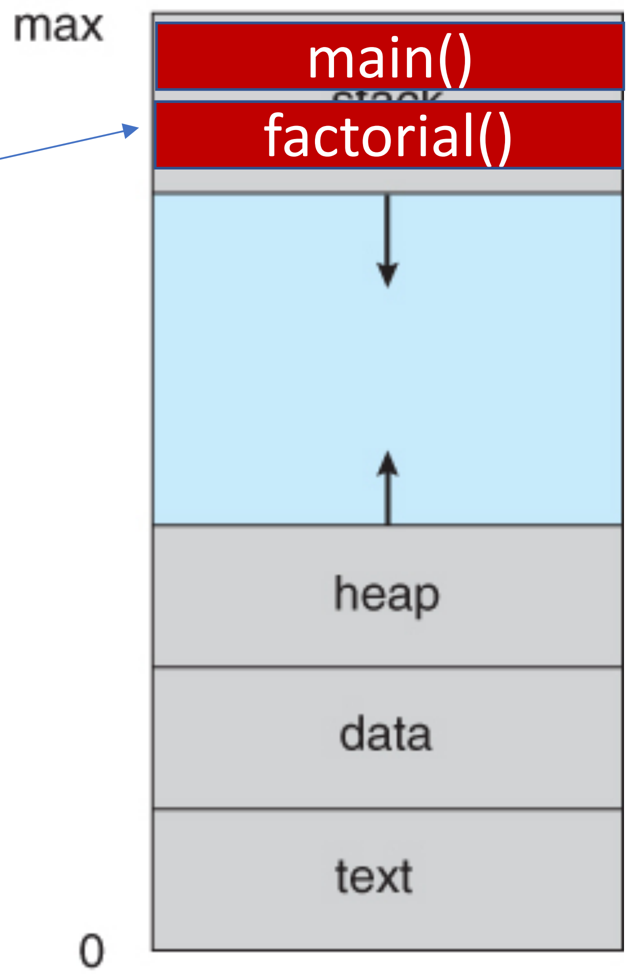
Activation Record



A Process in Memory

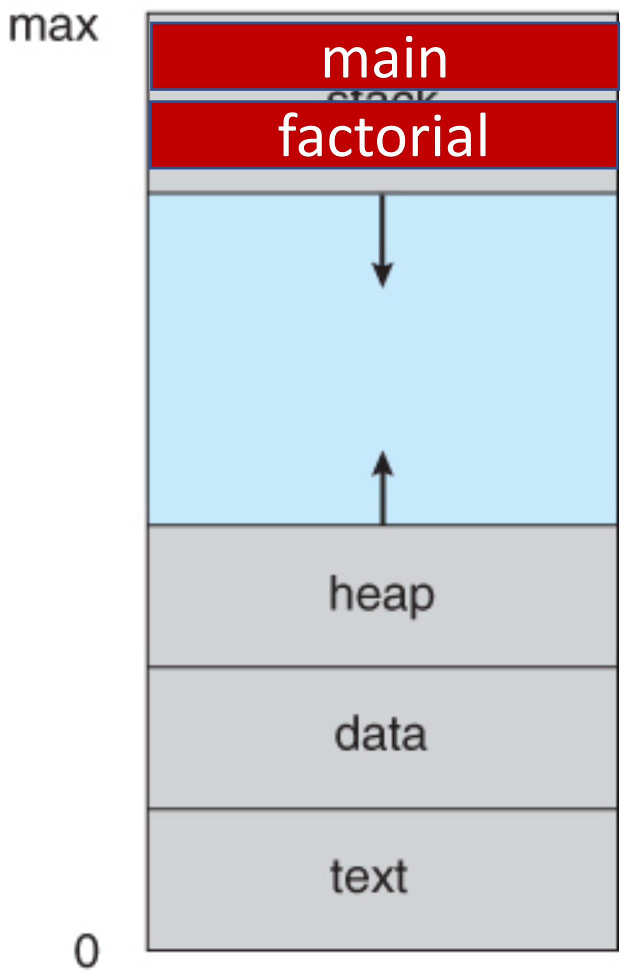
```
#include <stdio.h>
int factorial(int);
int n=6;
main(){
       int fac;
     fac = factorial(n);
       printf("The  factorial  is  %d",
  fac);
}

int factorial(int n){
       int fac = 1;
       while (n >= 1) {
              fac = fac * n;
              n = n-1;
```

Activation Record



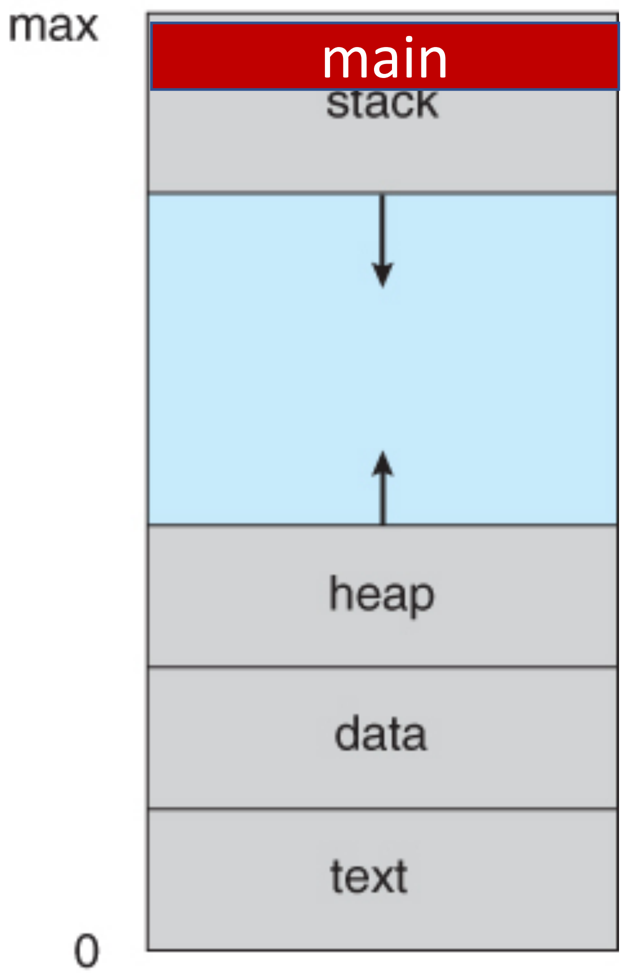
A Process in Memory

```
#include <stdio.h>
int factorial(int);
int n=6;
main(){
        int fac;
      fac = factorial(n);
        printf("The  factorial  is  %d",
   fac);
}

int factorial(int n){
        int fac = 1;
        while (n >= 1) {
                  fac = fac * n;
                  n = n-1;
```



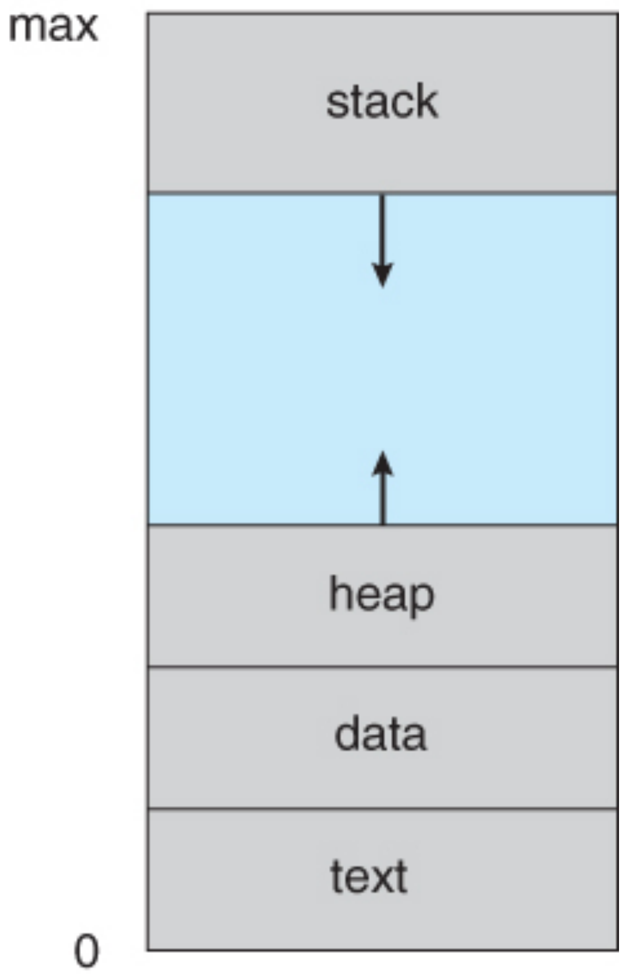
A Process in Memory

```
#include <stdio.h>
int factorial(int);
int n=6;
main(){
       int fac;
     fac = factorial(n);
       printf("The factorial is %d",
  fac);
}

int factorial(int n){
       int fac = 1;
       while (n >= 1) {
              fac = fac * n;
              n = n-1;
```



Space Complexity

Space Complexity is defined by the rate of growth of space in process space (stack, heap and data) with size of the problem.

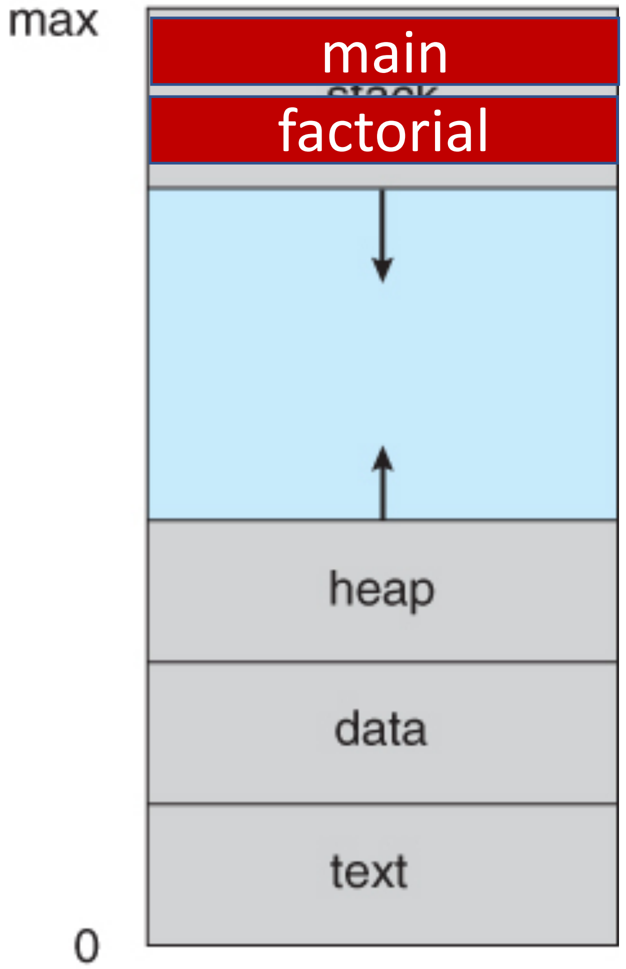


Space Complexity

Space Complexity is defined by the rate of growth of of space in process space (stack, heap and data) with size of the problem.
In the factorial example, only two activation records of constant size (independent of n) are stored in stack.

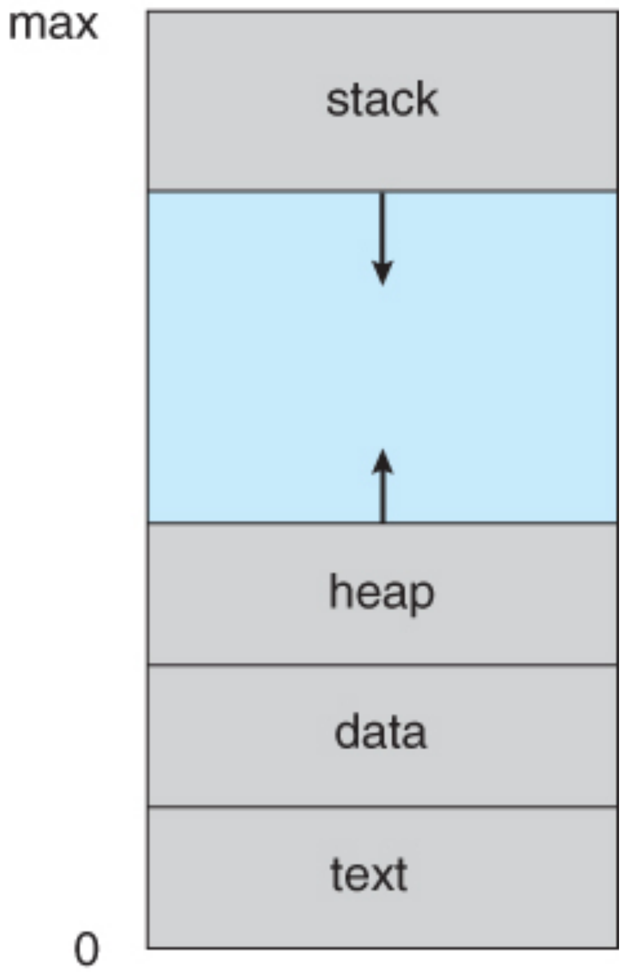
The heap and data sections are also constant (independent of n).

So, the space complexities for both the program and factorial function are constant i.e., $\theta(1)$



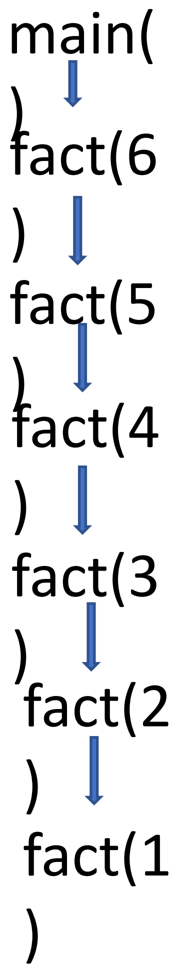
Space Complexity

```
#include <stdio.h>
int fact(int);
main(){
   int n = 6;
   printf(“The factorial is %d”, fact(n));
}
int fact(int n){
    if(n == 1)
    return 1;
    return n*fact(n-1);
}
```



Space Complexity

```
#include <stdio.h>
int fact(int);
main(){
   int n = 6;
   printf(“The factorial is %d”, fact(n));
}
int fact(int n){
    if(n == 1)
    return 1;
    return n*fact(n-1);
}
```



main()

fact(6)

fact(5)

fact(4)

fact(3)

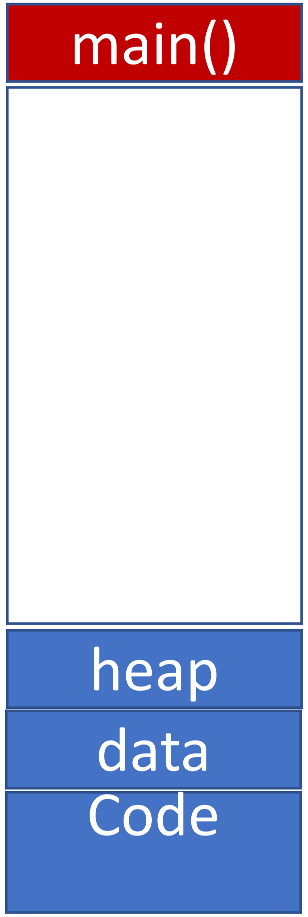
fact(2)

fact(1)

Space Complexity

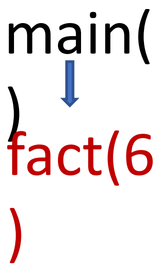
```
#include <stdio.h>
int fact(int);
main(){
   int n = 6;
   printf(“The factorial is %d”, fact(n));
}
int fact(int n){
    if(n == 1)
    return 1;
    return n*fact(n-1);
}
```

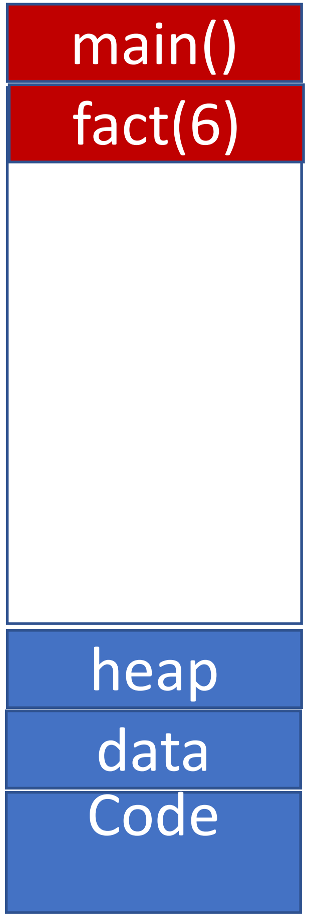
main(
)



Space Complexity

```
#include <stdio.h>
int fact(int);
main(){
   int n = 6;
   printf(“The factorial is %d”, fact(n));
}
int fact(int n){
    if(n == 1)
    return 1;
    return n*fact(n-1);
}
```

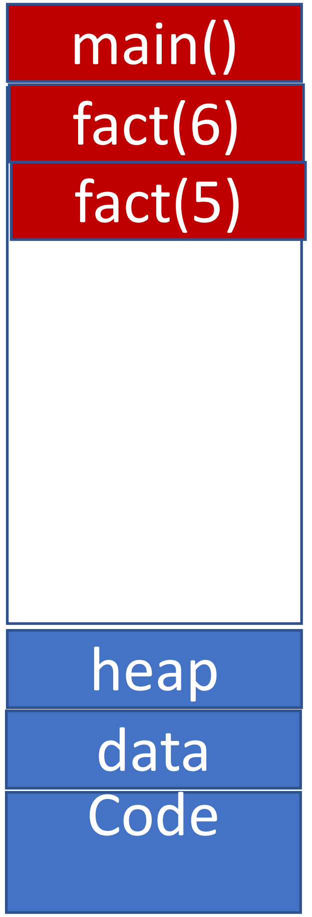




Space Complexity

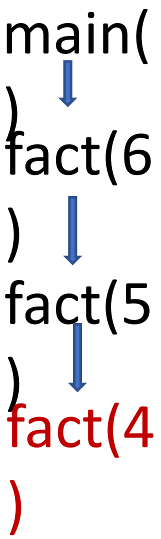
```
#include <stdio.h>
int fact(int);
main(){
   int n = 6;
   printf(“The factorial is %d”, fact(n));
}
int fact(int n){
    if(n == 1)
    return 1;
    return n*fact(n-1);
}
```

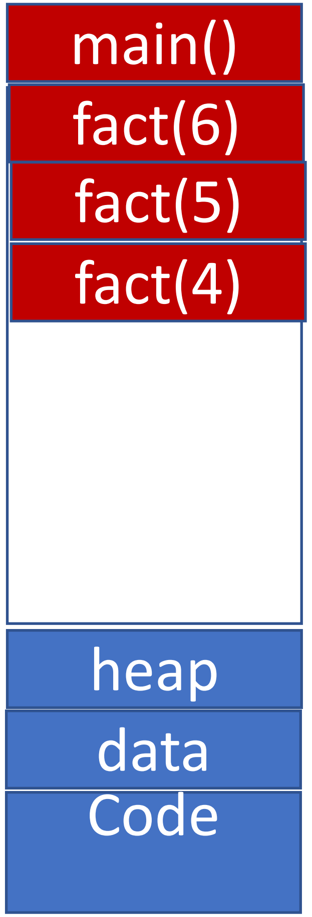
main()
↓
fact(6)
↓
fact(5)



Space Complexity

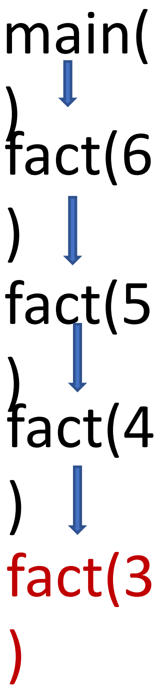
```
#include <stdio.h>
int fact(int);
main(){
   int n = 6;
   printf(“The factorial is %d”, fact(n));
}
int fact(int n){
    if(n == 1)
    return 1;
    return n*fact(n-1);
}
```





Space Complexity

```
#include <stdio.h>
int fact(int);
main(){
   int n = 6;
   printf(“The factorial is %d”, fact(n));
}
int fact(int n){
    if(n == 1)
    return 1;
    return n*fact(n-1);
}
```



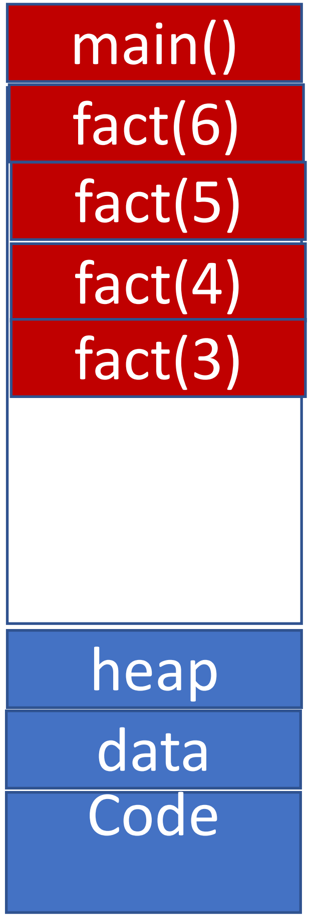
main( )

fact(6)

fact(5)

fact(4)

fact(3)



main()

fact(6)

fact(5)

fact(4)

fact(3)

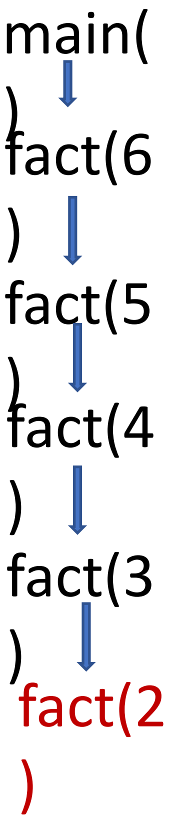
heap

data

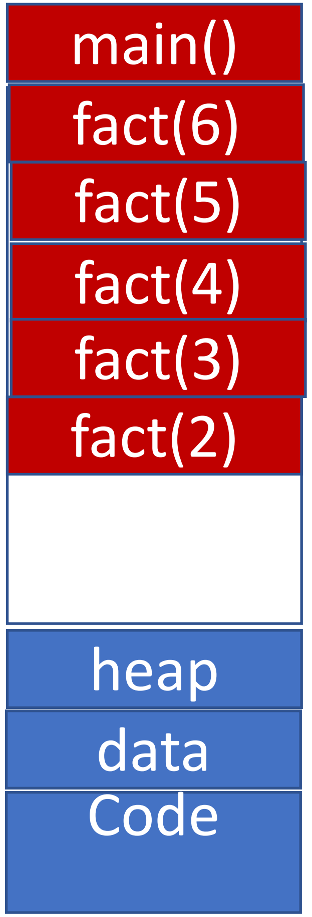
Code

Space Complexity

```
#include <stdio.h>
int fact(int);
main(){
   int n = 6;
   printf(“The factorial is %d”, fact(n));
}
int fact(int n){
    if(n == 1)
    return 1;
    return n*fact(n-1);
}
```



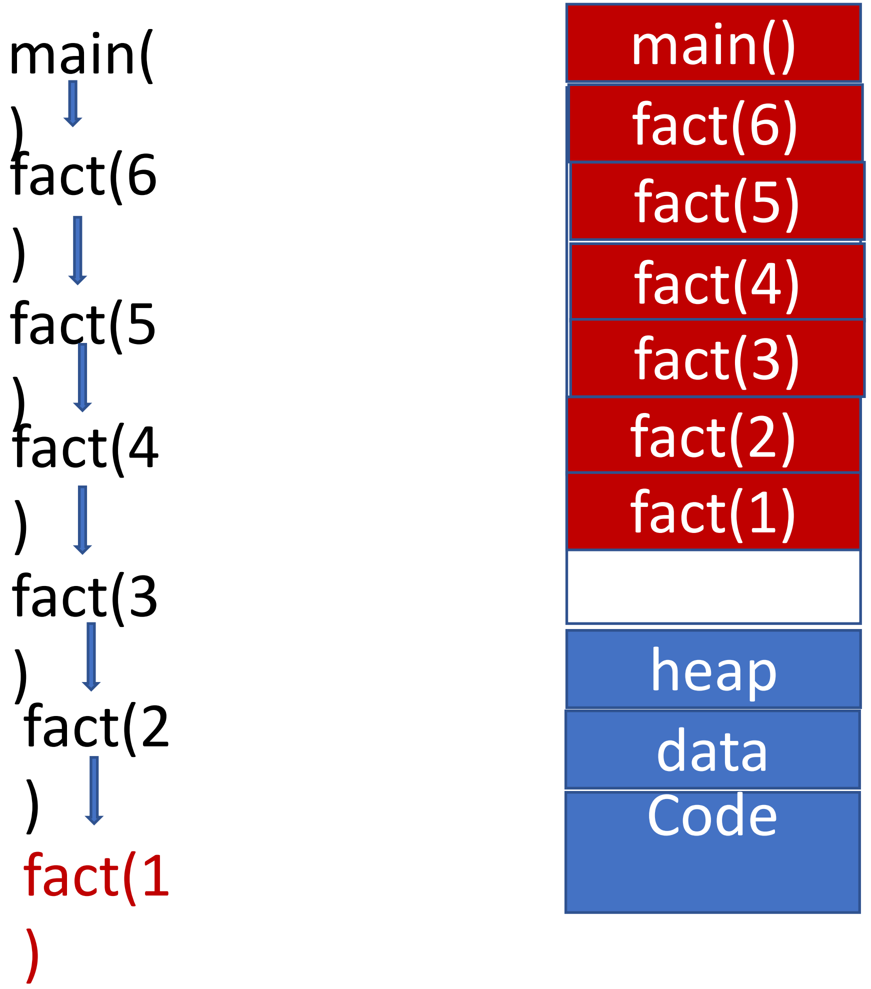
main(
)
fact(6
)
fact(5
)
fact(4
)
fact(3
)
fact(2
)



main()
fact(6)
fact(5)
fact(4)
fact(3)
fact(2)
heap
data
Code

Space Complexity

```
#include <stdio.h>
int fact(int);
main(){
   int n = 6;
   printf(“The factorial is %d”, fact(n));
}
int fact(int n){
    if(n == 1)
    return 1;
    return n*fact(n-1);
}
```



main()

fact(6)

fact(5)

fact(4)

fact(3)

fact(2)

fact(1)

main()

fact(6)

fact(5)

fact(4)

fact(3)

fact(2)

fact(1)

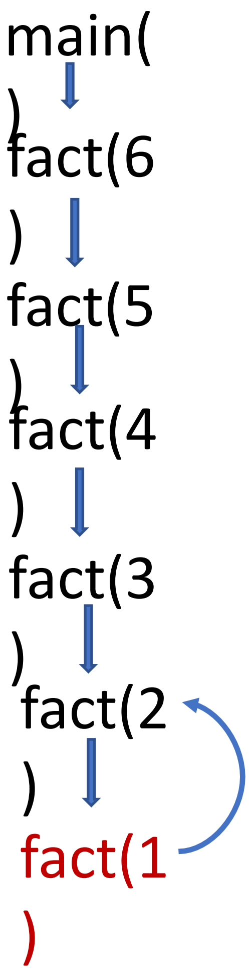
heap

data

Code

Space Complexity

```
#include <stdio.h>
int fact(int);
main(){
   int n = 6;
   printf(“The factorial is %d”, fact(n));
}
int fact(int n){
    if(n == 1)
   return 1;
    return n*fact(n-1);
}
```


main()

fact(6)

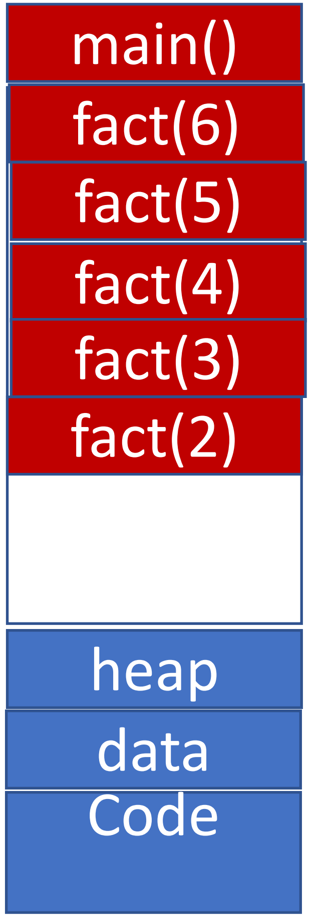
fact(5)

fact(4)

fact(3)

fact(2)

fact(1)


main()

fact(6)

fact(5)

fact(4)

fact(3)

fact(2)

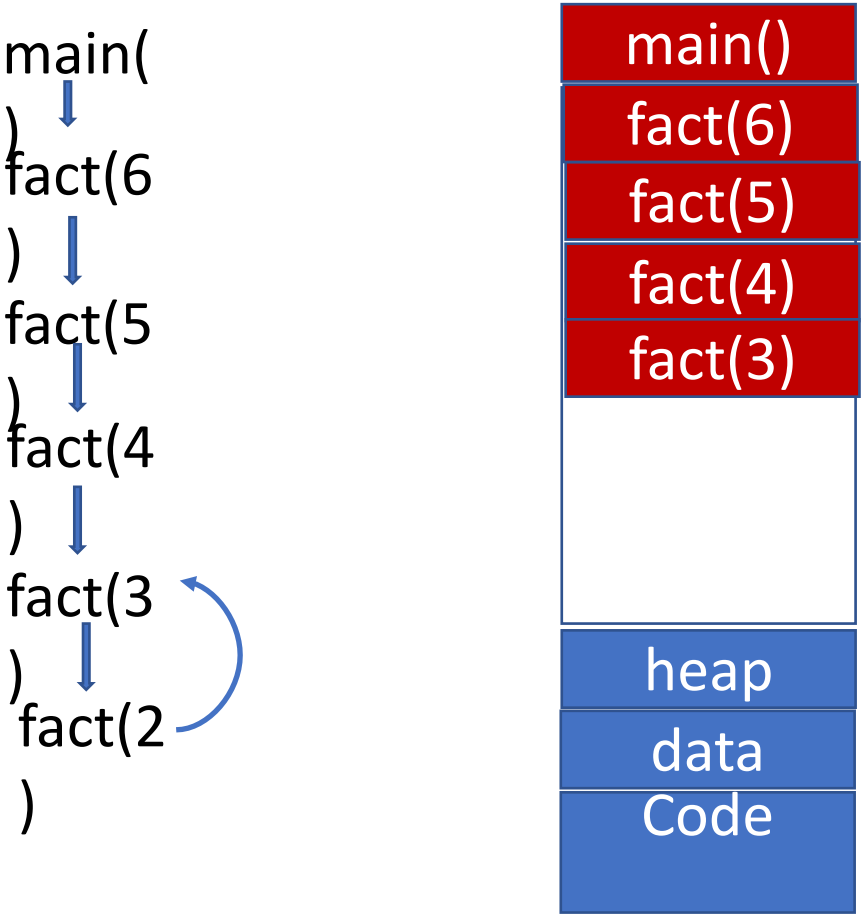
heap

data

Code

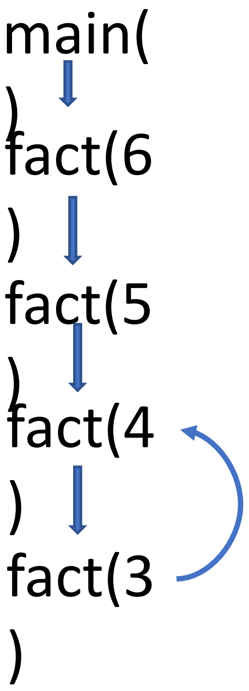
Space Complexity

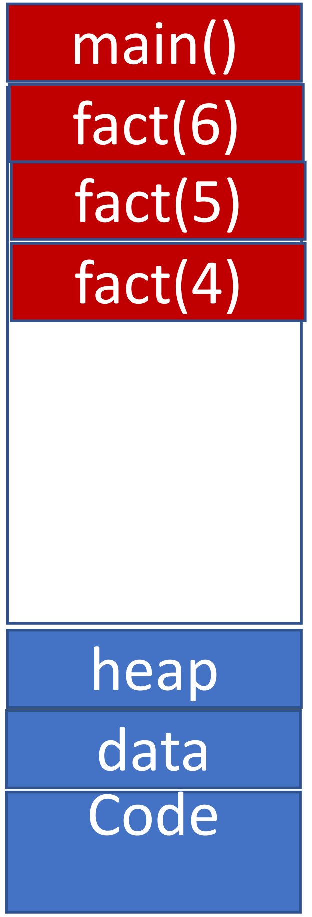
```
#include <stdio.h>
int fact(int);
main(){
   int n = 6;
   printf("The factorial is %d", fact(n));
}
int fact(int n){
    if(n == 1)
   return 1;
    return n*fact(n-1);
}
```



Space Complexity

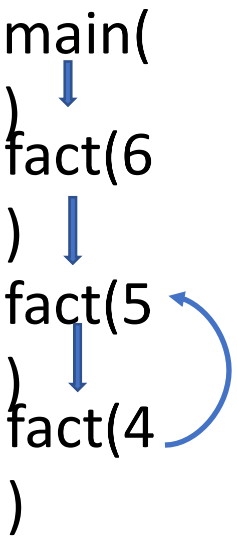
```
#include <stdio.h>
int fact(int);
main(){
   int n = 6;
   printf(“The factorial is %d”, fact(n));
}
int fact(int n){
    if(n == 1)
   return 1;
    return n*fact(n-1);
}
```

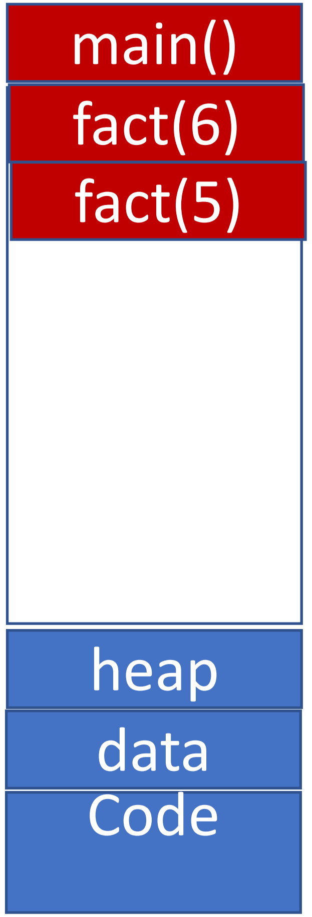




Space Complexity

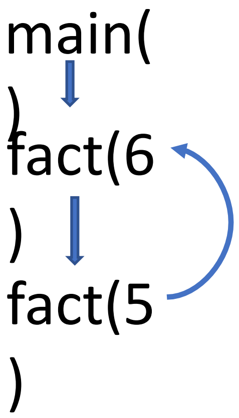
```
#include <stdio.h>
int fact(int);
main(){
   int n = 6;
   printf("The factorial is %d", fact(n));
}
int fact(int n){
    if(n == 1)
   return 1;
    return n*fact(n-1);
}
```





Space Complexity

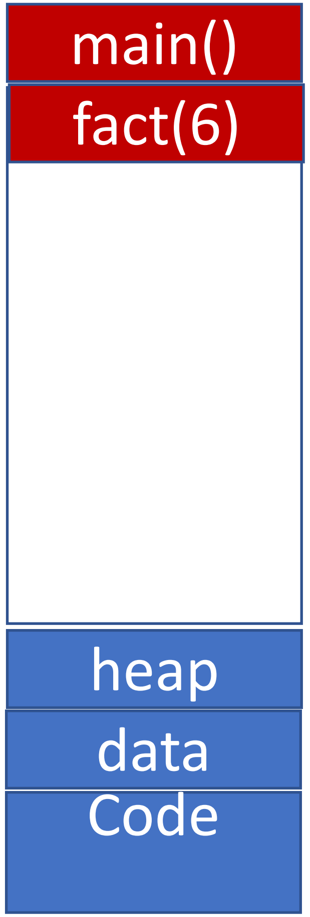
```
#include <stdio.h>
int fact(int);
main(){
   int n = 6;
   printf(“The factorial is %d”, fact(n));
}
int fact(int n){
    if(n == 1)
   return 1;
    return n*fact(n-1);
}
```



main()

fact(6)

fact(5)



main()

fact(6)

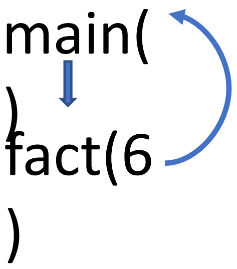
heap

data

Code

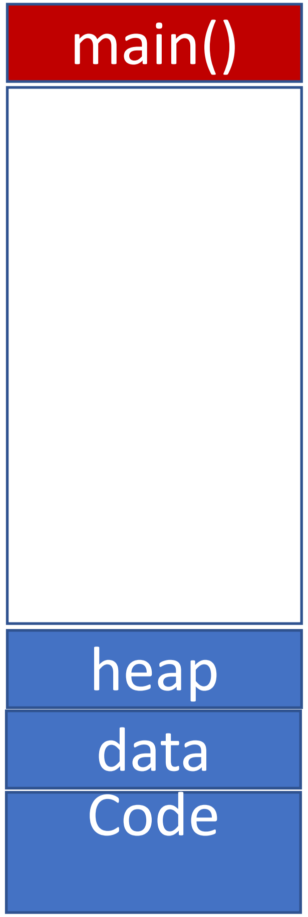
Space Complexity

```
#include <stdio.h>
int fact(int);
main(){
   int n = 6;
   printf(“The factorial is %d”, fact(n));
}
int fact(int n){
    if(n == 1)
   return 1;
    return n*fact(n-1);
}
```



main()

fact(6)



main()

heap

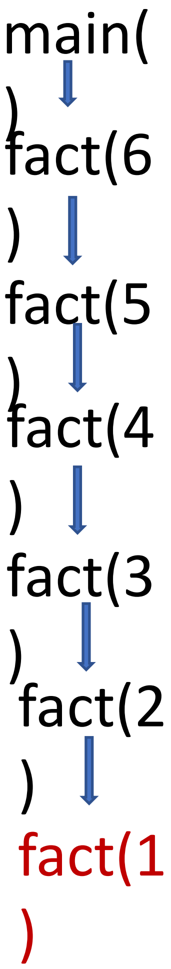
data

Code

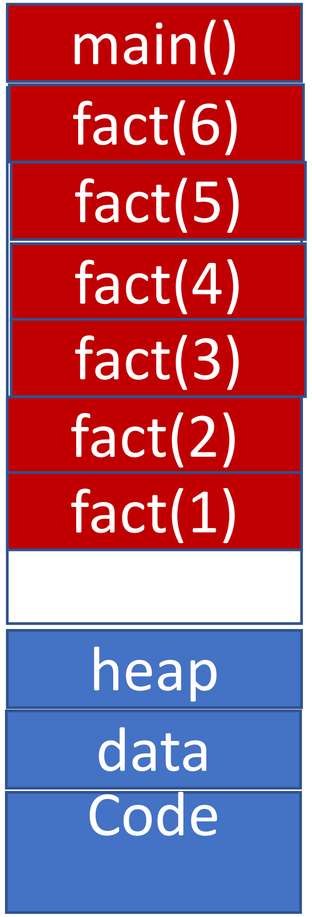
Space Complexity

```
#include <stdio.h>
int fact(int);
main(){
   int n = 6;
   printf(“The factorial is %d”, fact(n));
}
int fact(int n){
    if(n == 1)
    return 1;
    return n*fact(n-1);
}
```

Space complexity: $\theta(n)$



main()
fact(6)
fact(5)
fact(4)
fact(3)
fact(2)
fact(1)



main()
fact(6)
fact(5)
fact(4)
fact(3)
fact(2)
fact(1)
heap
data
Code