GNG1106
Fundamentals of Engineering Computation

Instructor: **Hitham Jleed**

University of Ottawa

Fall 2023 ~

In-Class Exercise:

Outline

1. Dynamic Memory Allocation (NOT TESTED)

- For variables and arrays, the amount of memory (i.e., the number of bytes) to be allocated needs to be known at compiling time.
- Dynamic memory allocation refers to allocating memory at run time where the amount of allocated memory is resolved during the program execution.
- Dynamic memory allocation is achieved by calling function `malloc` (and its relatives).
- Using `malloc` requires `#include <stdlib.h>`

- The parameter passed into `malloc` is the amount of memory to be allocated in unit of bytes.
- When calling `malloc`, if the requested memory allocation is successful, the function returns the address (of the first byte) of the memory block allocated.
- If the requested memory allocation fails, `malloc` returns `NULL`.
- Function `malloc` is ignorant about the type of data that will be stored in the memory, and casting the returned value of `malloc` (i.e., an address) to the right pointer type is required.
- To release a block of dynamically allocated memory, pass the pointer to the memory to the `free` function.

```
#include <stdlib.h> // you need this to call
malloc
...
int N;
int *p=NULL;

... // obtain the value of N

p=(int *)malloc(N*sizeof(int));

if (p!=NULL)
{
... // use the memory pointed to by p
}
free(p);
```

- Memory allocated using `malloc` resides in heap and can be legally accessed by any function.
    - Recall that an array/variable declared in a function is only accessible from within the function.
- The life time of memory allocated by `malloc` is ended only when its address is passed to the `free` function.
    - Recall that an array/variable declared in a function has its life time ended when the function exits.

Watch out for Memory Leak!

- Memory leak refers to the scenario in which some (dynamically) allocated memory has not been released (i.e. not “free-ed”) and yet it is no longer accessible by the program.
- If a block memory is allocated by `malloc` inside a function, then the memory should either be released (by calling `free`) or have its address returned to outside the function. Otherwise memory leak will occur!
- Compiler is not able to detect memory leak!

Highlight

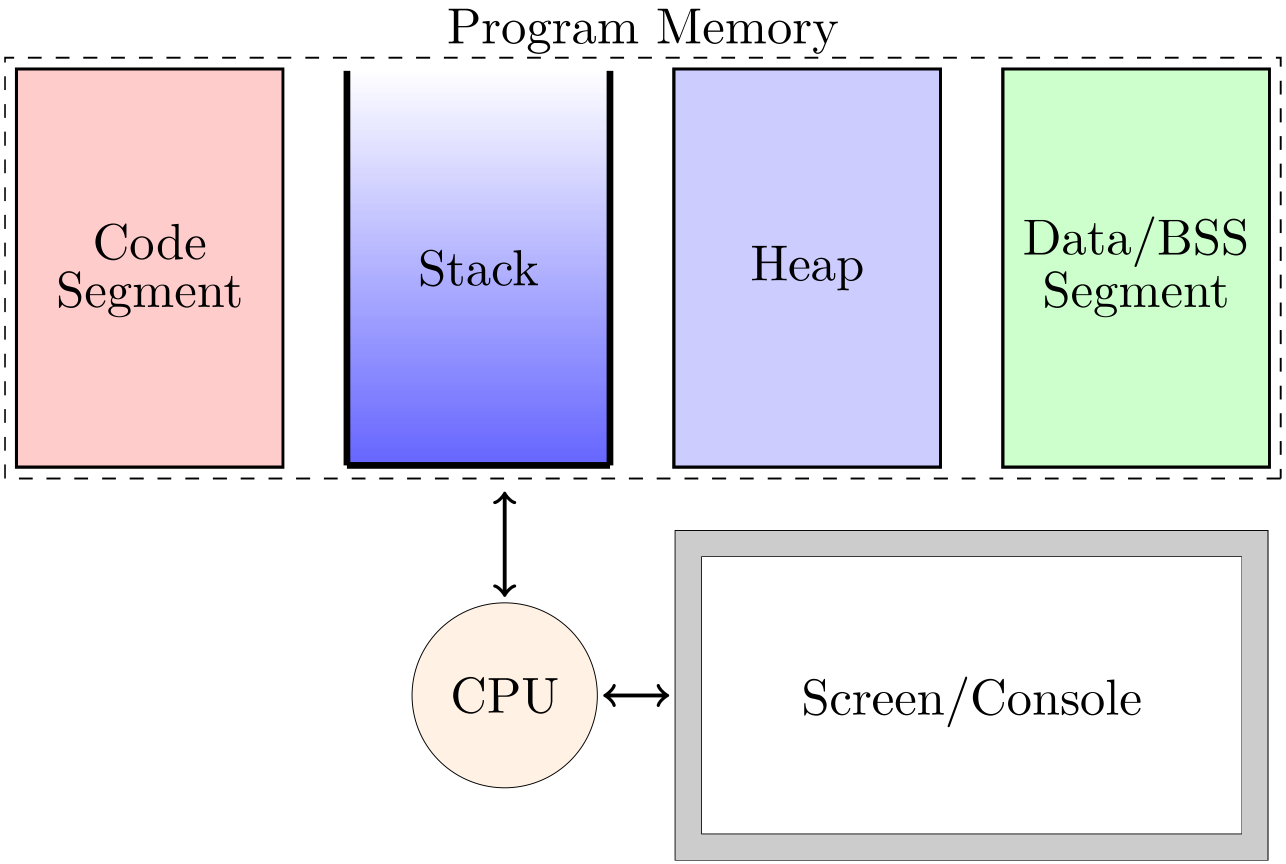
For every call of `malloc`, ask yourself: *who (i.e., which part of the program) will be responsible for free-ing it?*

- Write a function that takes an array as input parameter and returns the median value of the array. Note: after the function call, the array passed to the function can not be changed.

The findMedian Program (Simplified)

```
void sort(int *ptr, int len)
{
           ...
}
int median(int *ptr, int len)
{
  int i, out, *p;
  p=(int *)malloc(len*sizeof(int));
  for (i=0; i<len; i++)
            p[i]=ptr[i];
  sort(p, len);
  out=p[len/2];
  free(p);
  return out;
}
int main()
{
  int x[5]={4, 8, 2, 1, 9};
  printf("%d\n",median(x, 5));
  return 0;
}
```

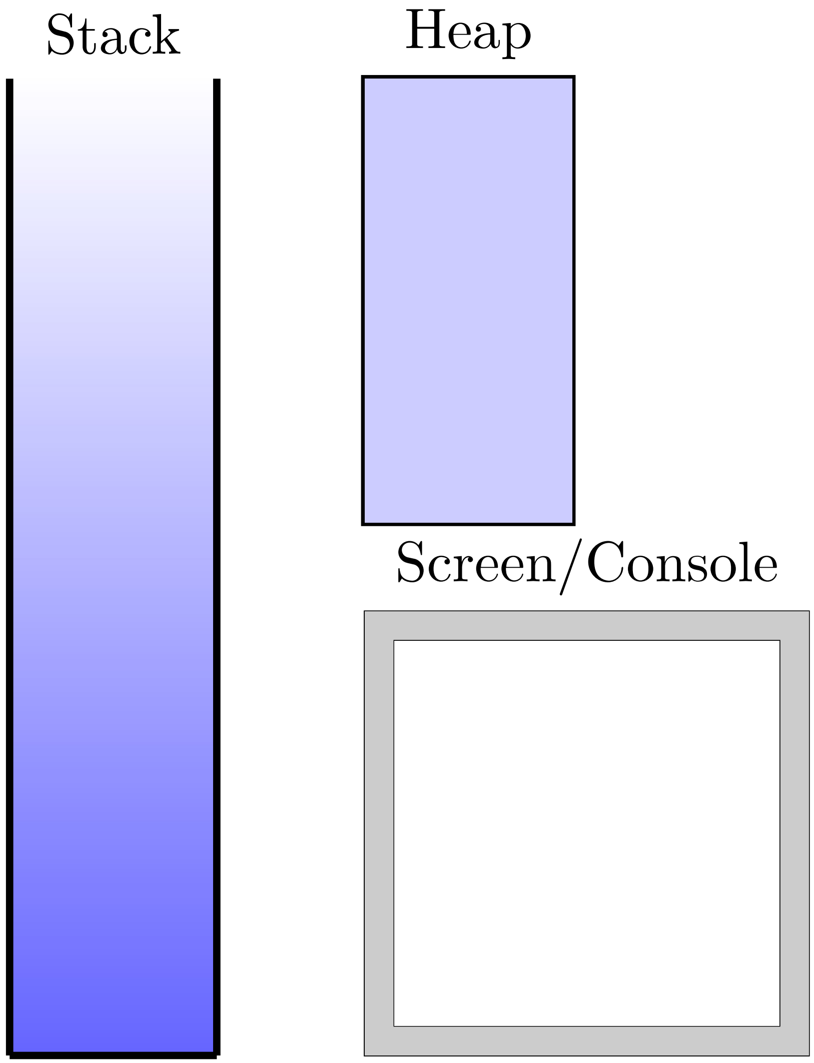
“Programming Model” used in This Course



Trace findMedian Program

Code Segment

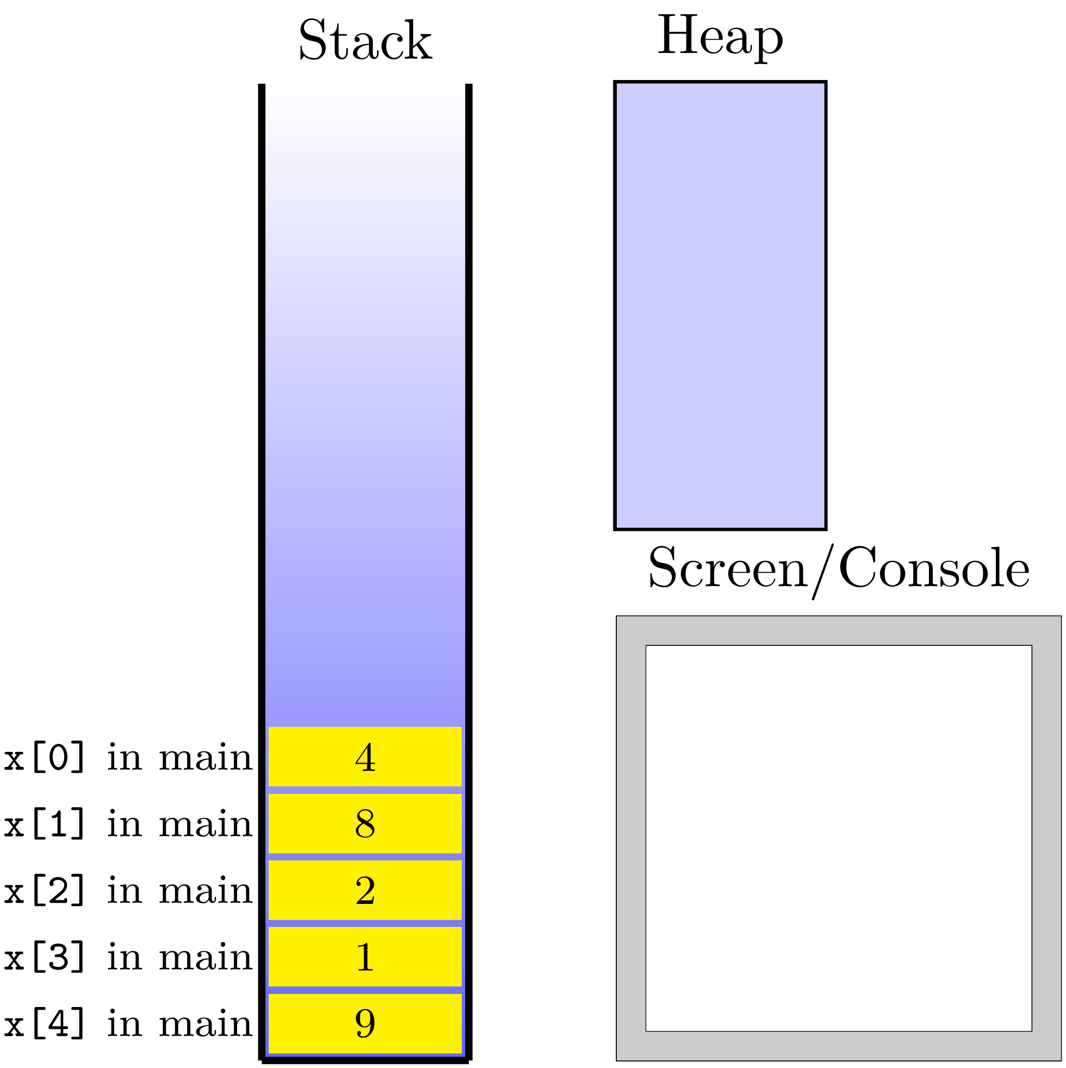
```
void sort(int *ptr, int len)
{              ...              }
int median(int *ptr, int len)
{
   int i, out, *p;
   p=(int *)malloc(len*
               sizeof(int));
   for (i=0; i<len; i++)
               p[i]=ptr[i];
   sort(p, len);
   out=p[len/2];
   free(p);
   return out;
}
int main()
{
   int x[5]={4, 8, 2, 1, 9};
   printf("%d\n",median(x, 5));
   return 0;
}
```



Trace findMedian Program (1)

Code Segment

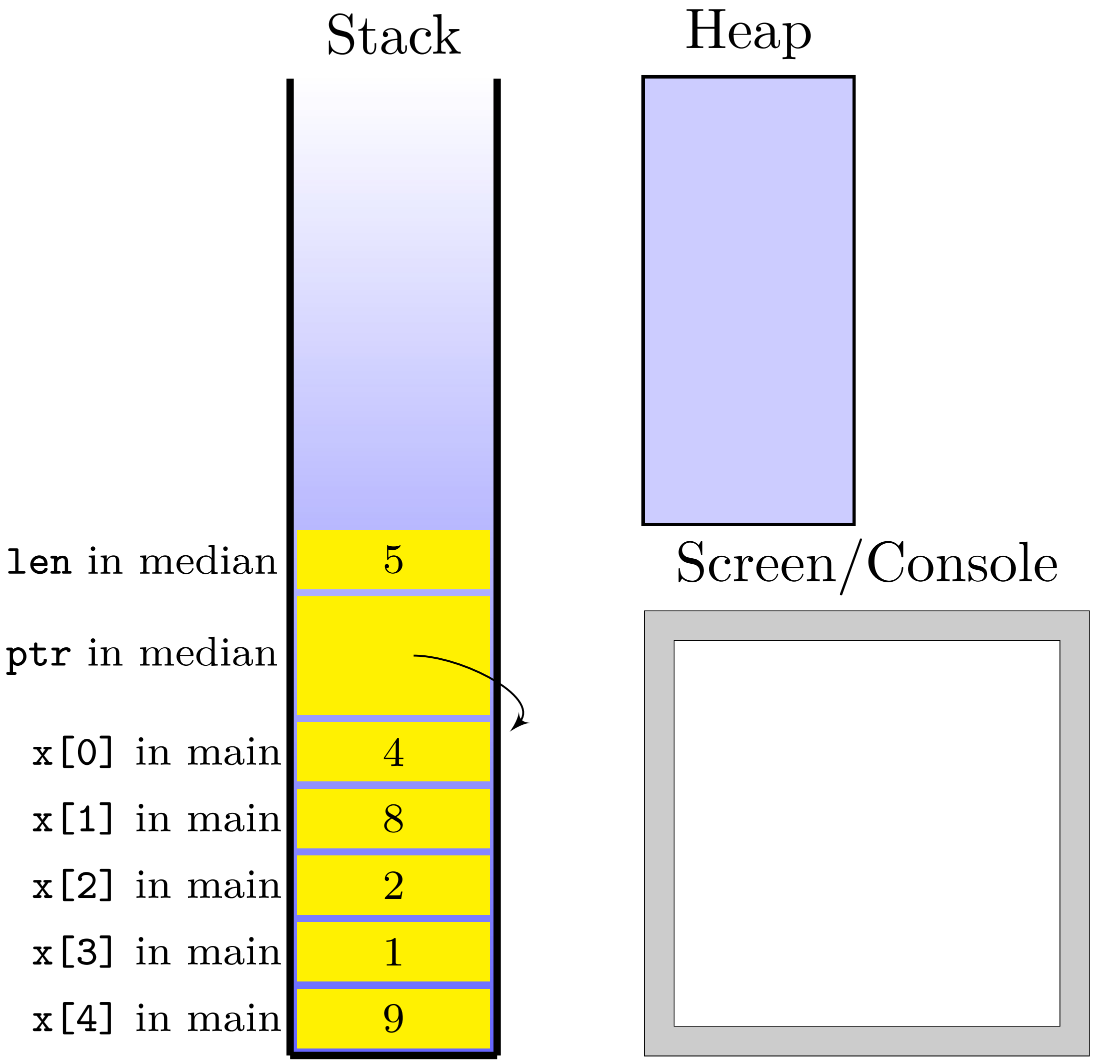
```
void sort(int *ptr, int len)
{            ...              }
int median(int *ptr, int len)
{
   int i, out, *p;
   p=(int *)malloc(len*
               sizeof(int));
   for (i=0; i<len; i++)
               p[i]=ptr[i];
   sort(p, len);
   out=p[len/2];
   free(p);
   return out;
}
int main()
{
   int x[5]={4, 8, 2, 1, 9};
   printf("%d\n",median(x, 5));
   return 0;
}
```



Trace findMedian Program (2)

Code Segment

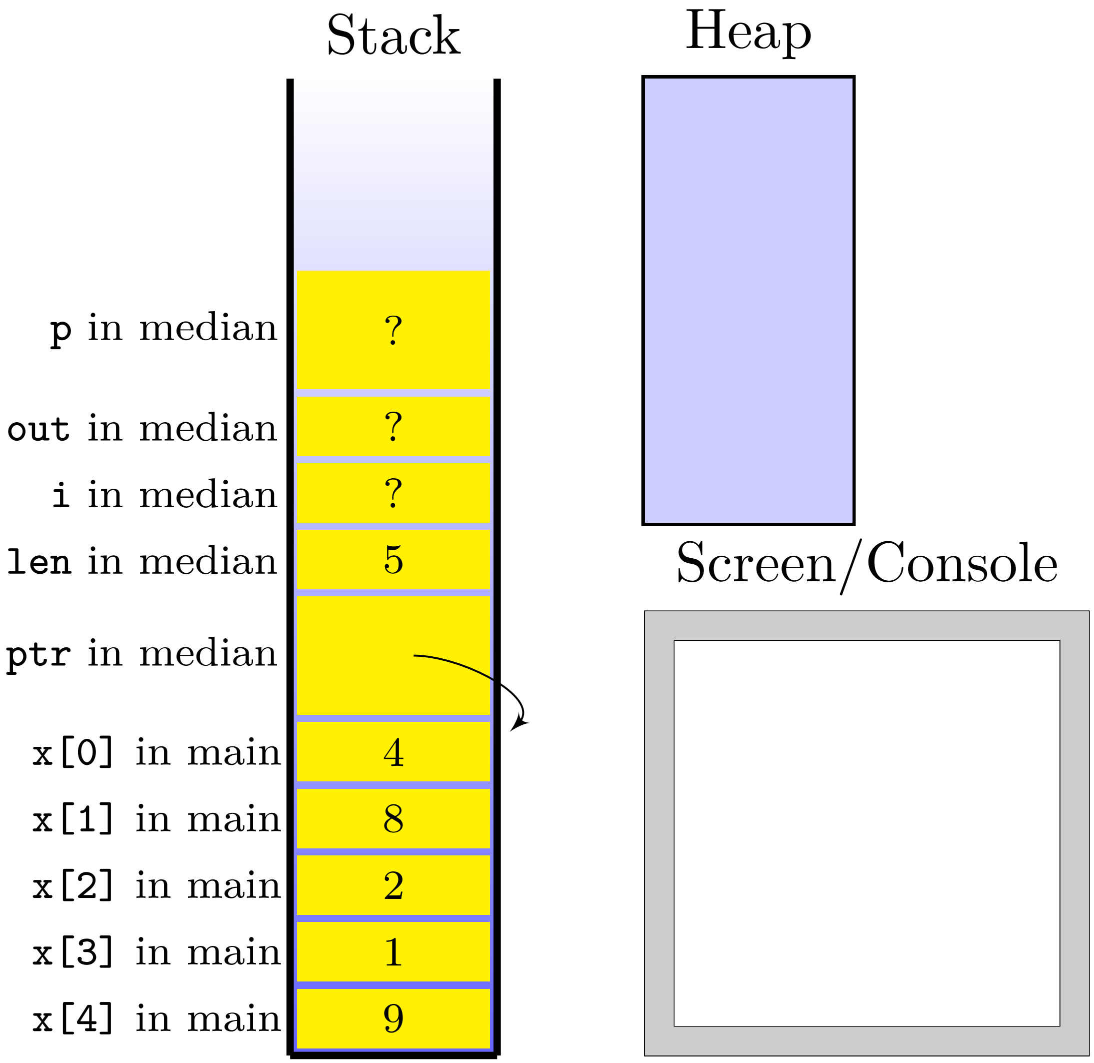
```
void sort(int *ptr, int len)
{            ...             }
int median(int *ptr, int len)
{
  int i, out, *p;
  p=(int *)malloc(len*
           sizeof(int));
  for (i=0; i<len; i++)
           p[i]=ptr[i];
  sort(p, len);
  out=p[len/2];
  free(p);
  return out;
}
int main()
{
  int x[5]={4, 8, 2, 1, 9};
  printf("%d\n",median(x, 5));
  return 0;
}
```



Trace findMedian Program (3)

Code Segment

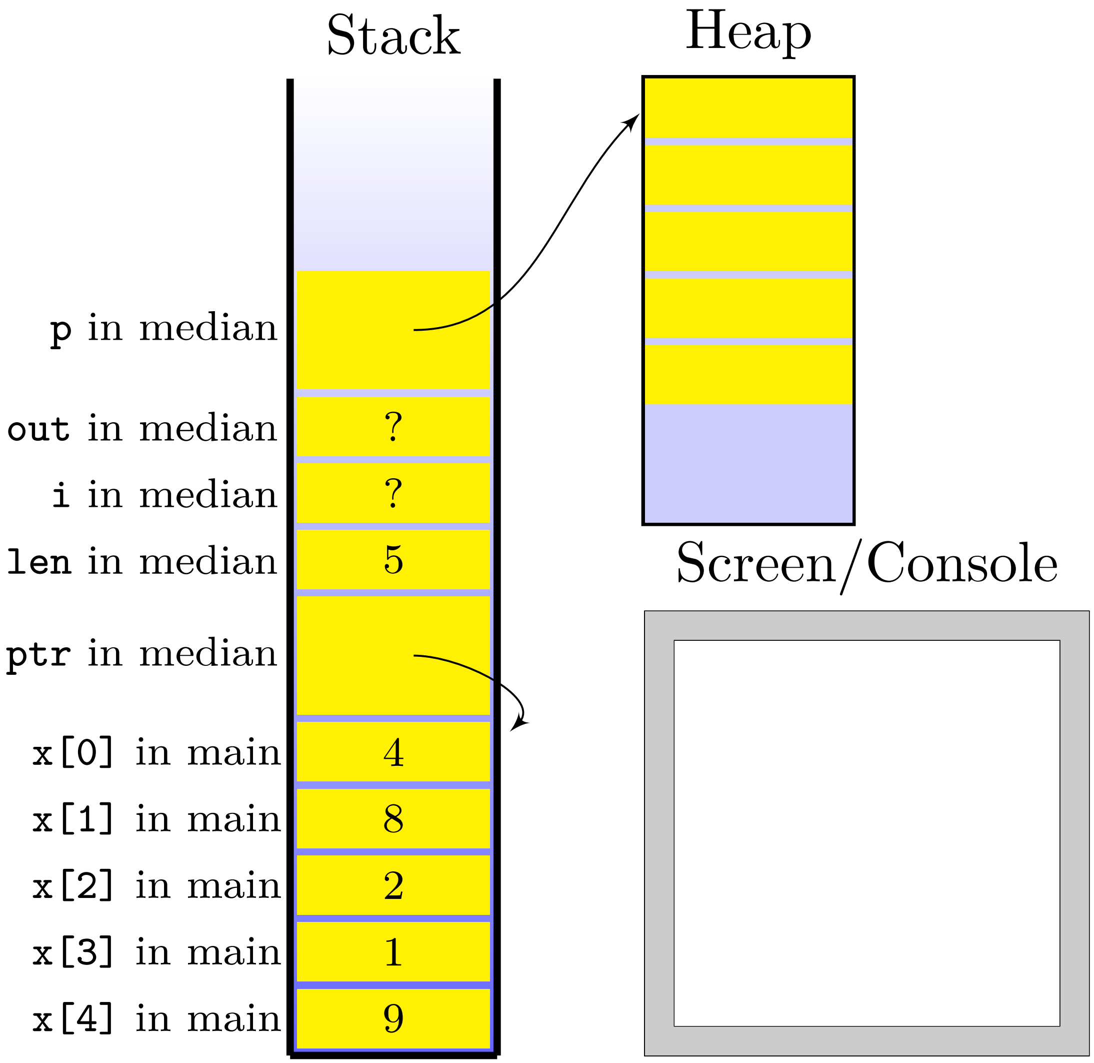
```
void sort(int *ptr, int len)
{            ...            }
int median(int *ptr, int len)
{
   int i, out, *p;
   p=(int *)malloc(len*
            sizeof(int));
   for (i=0; i<len; i++)
            p[i]=ptr[i];
   sort(p, len);
   out=p[len/2];
   free(p);
   return out;
}
int main()
{
   int x[5]={4, 8, 2, 1, 9};
   printf("%d\n",median(x, 5));
   return 0;
}
```



Trace findMedian Program (4)

Code Segment

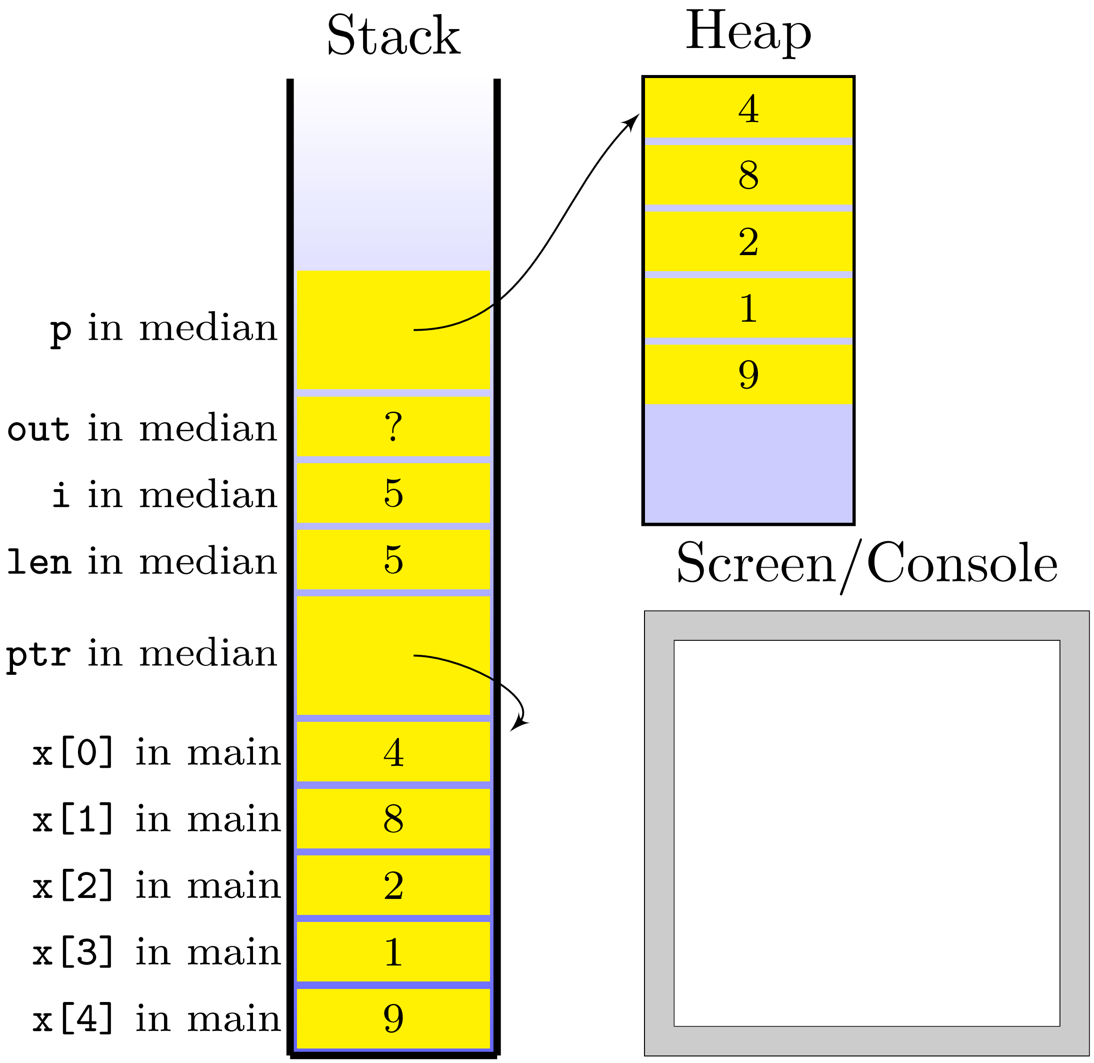
```
void sort(int *ptr, int len)
{            ...             }
int median(int *ptr, int len)
{
   int i, out, *p;
   p=(int *)malloc(len*
            sizeof(int));
   for (i=0; i<len; i++)
            p[i]=ptr[i];
   sort(p, len);
   out=p[len/2];
   free(p);
   return out;
}
int main()
{
   int x[5]={4, 8, 2, 1, 9};
   printf("%d\n",median(x, 5));
   return 0;
}
```



Trace findMedian Program (5)

Code Segment

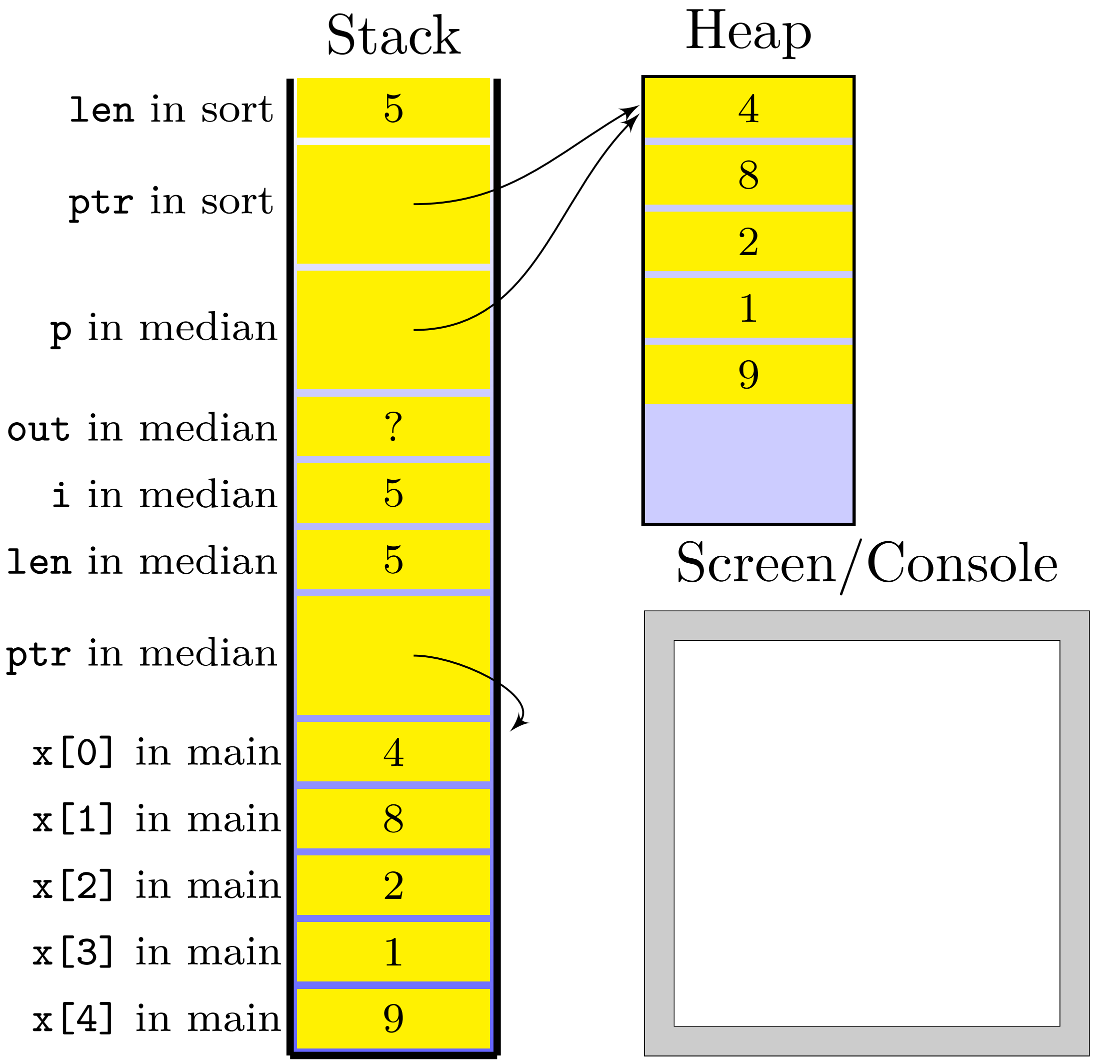
```
void sort(int *ptr, int len)
{            ...              }
int median(int *ptr, int len)
{
   int i, out, *p;
   p=(int *)malloc(len*
            sizeof(int));
   for (i=0; i<len; i++)
            p[i]=ptr[i];
   sort(p, len);
   out=p[len/2];
   free(p);
   return out;
}
int main()
{
   int x[5]={4, 8, 2, 1, 9};
   printf("%d\n",median(x, 5));
   return 0;
}
```



Trace findMedian Program (6)

Code Segment

```
void sort(int *ptr, int len)
{            ...            }
int median(int *ptr, int len)
{
   int i, out, *p;
   p=(int *)malloc(len*
              sizeof(int));
   for (i=0; i<len; i++)
              p[i]=ptr[i];
   sort(p, len);
   out=p[len/2];
   free(p);
   return out;
}
int main()
{
   int x[5]={4, 8, 2, 1, 9};
   printf("%d\n",median(x, 5));
   return 0;
}
```



Stack

len in sort 5

ptr in sort

p in median

out in median ?

i in median 5

len in median 5

ptr in median

x[0] in main 4

x[1] in main 8

x[2] in main 2

x[3] in main 1

x[4] in main 9

Heap

4

8

2

1

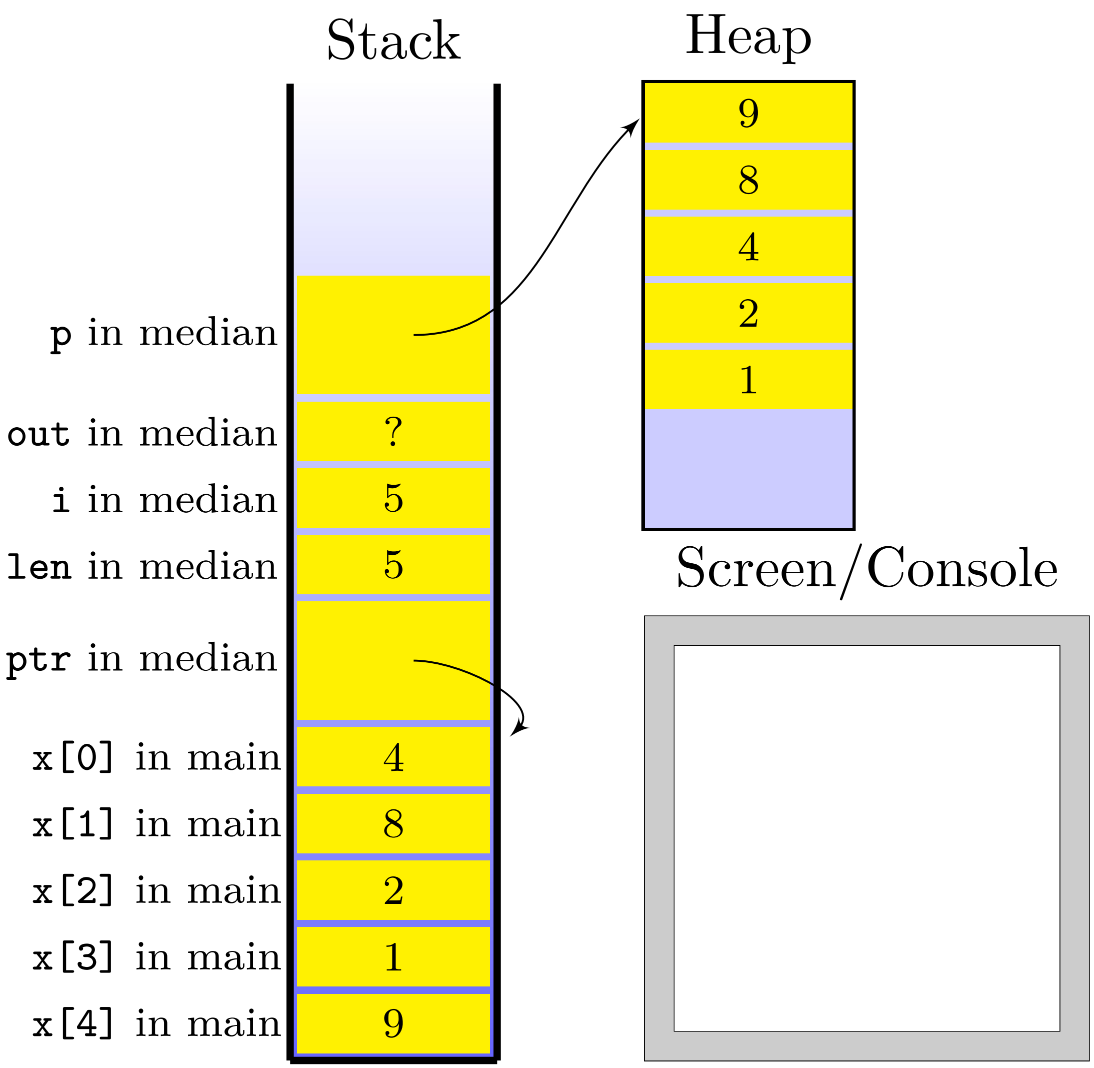
9

Screen/Console

Trace findMedian Program (7)

Code Segment

```
void sort(int *ptr, int len)
{            ...            }
int median(int *ptr, int len)
{
   int i, out, *p;
   p=(int *)malloc(len*
            sizeof(int));
   for (i=0; i<len; i++)
            p[i]=ptr[i];
   sort(p, len);
   out=p[len/2];
   free(p);
   return out;
}
int main()
{
   int x[5]={4, 8, 2, 1, 9};
   printf("%d\n",median(x, 5));
   return 0;
}
```



Stack

p in median

out in median ?

i in median 5

len in median 5

ptr in median

x[0] in main 4

x[1] in main 8

x[2] in main 2

x[3] in main 1

x[4] in main 9

Heap

9

8

4

2

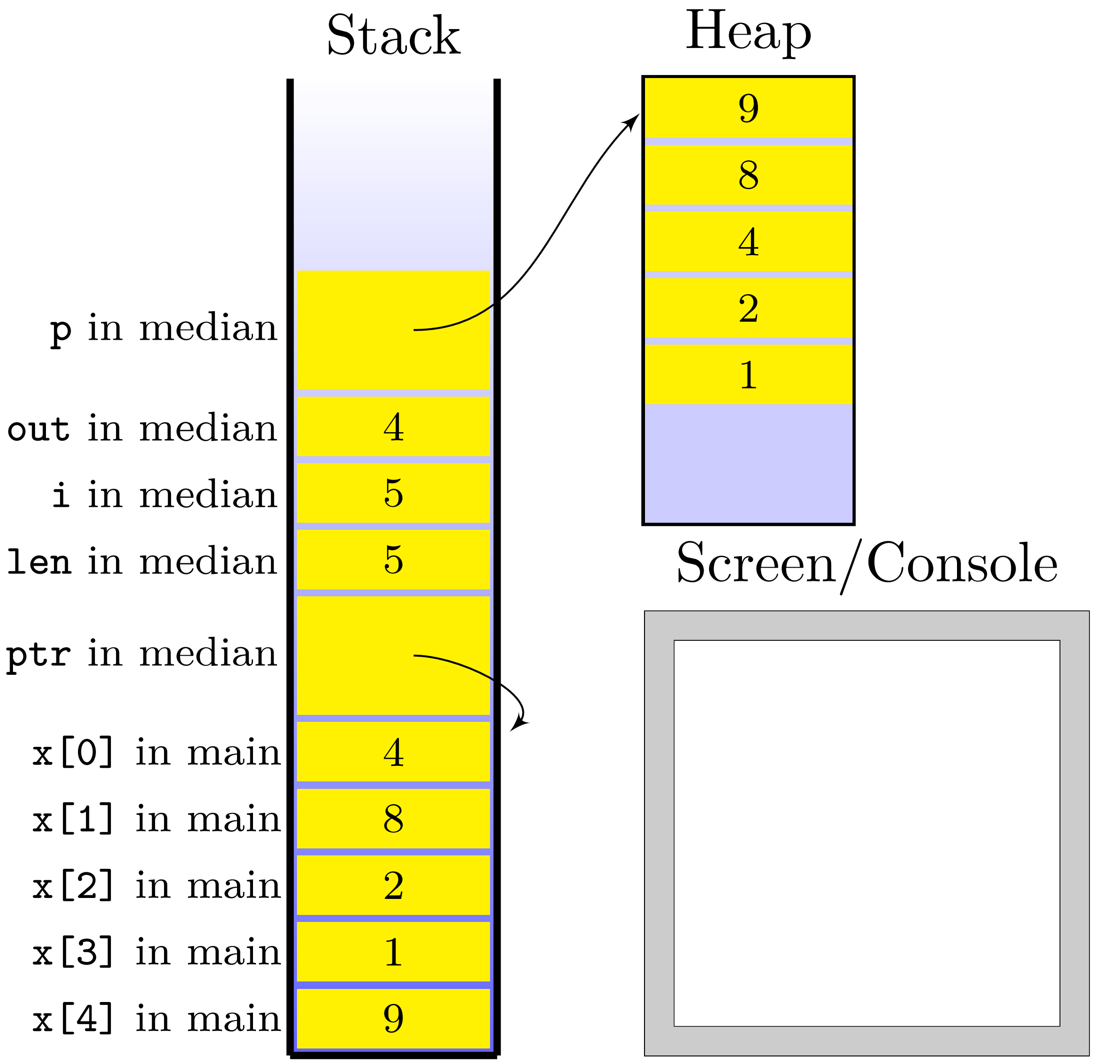
1

Screen/Console

Trace findMedian Program (8)

Code Segment

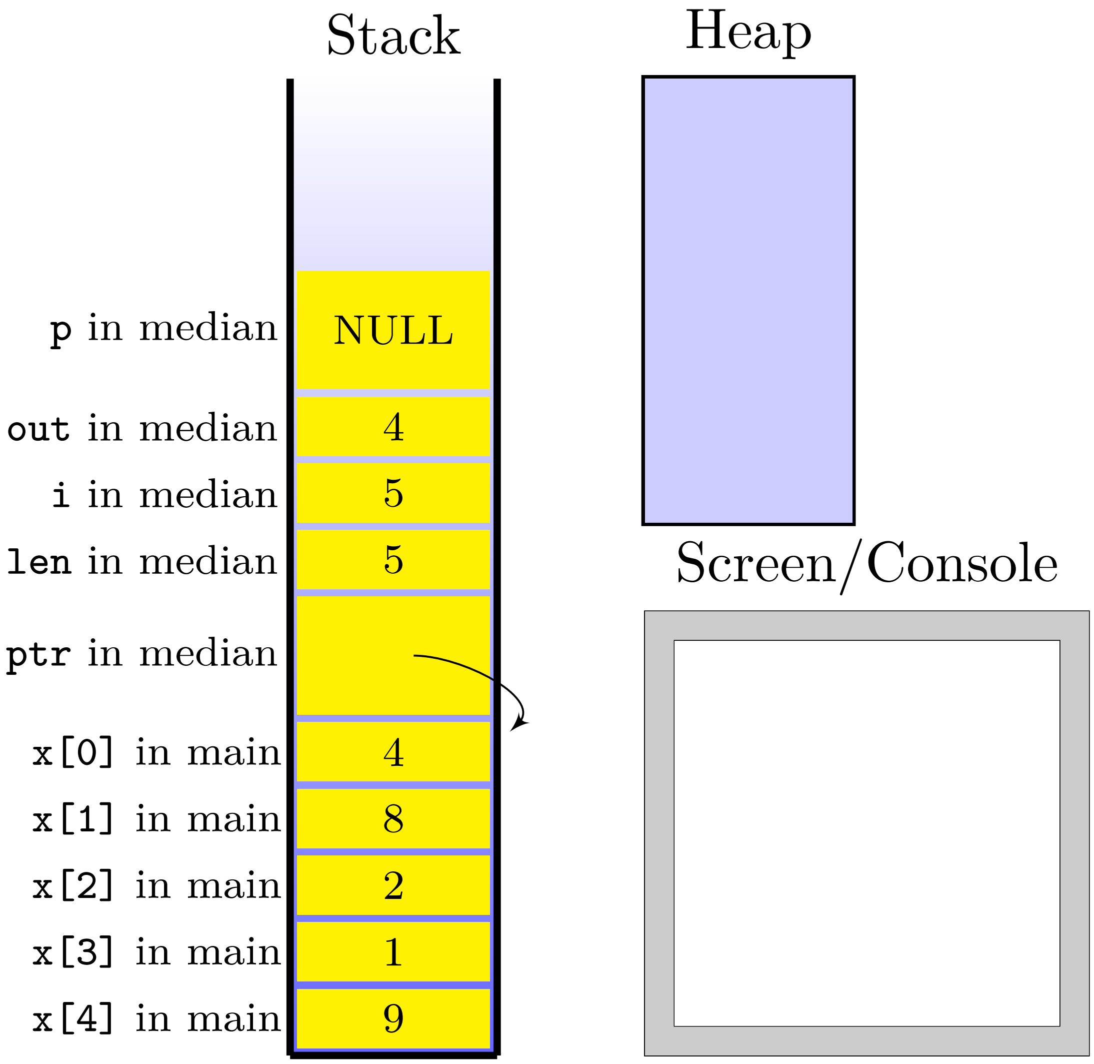
```
void sort(int *ptr, int len)
{           ...            }
int median(int *ptr, int len)
{
   int i, out, *p;
   p=(int *)malloc(len*
            sizeof(int));
   for (i=0; i<len; i++)
            p[i]=ptr[i];
   sort(p, len);
   out=p[len/2];
   free(p);
   return out;
}
int main()
{
   int x[5]={4, 8, 2, 1, 9};
   printf("%d\n",median(x, 5));
   return 0;
}
```



Trace findMedian Program (9)

Code Segment

```
void sort(int *ptr, int len)
{            ...              }
int median(int *ptr, int len)
{
   int i, out, *p;
   p=(int *)malloc(len*
               sizeof(int));
   for (i=0; i<len; i++)
               p[i]=ptr[i];
   sort(p, len);
   out=p[len/2];
   free(p);
   return out;
}
int main()
{
   int x[5]={4, 8, 2, 1, 9};
   printf("%d\n",median(x, 5));
   return 0;
}
```



Stack

p in median	NULL
out in median	4
i in median	5
len in median	5
ptr in median	
x[0] in main	4
x[1] in main	8
x[2] in main	2
x[3] in main	1
x[4] in main	9

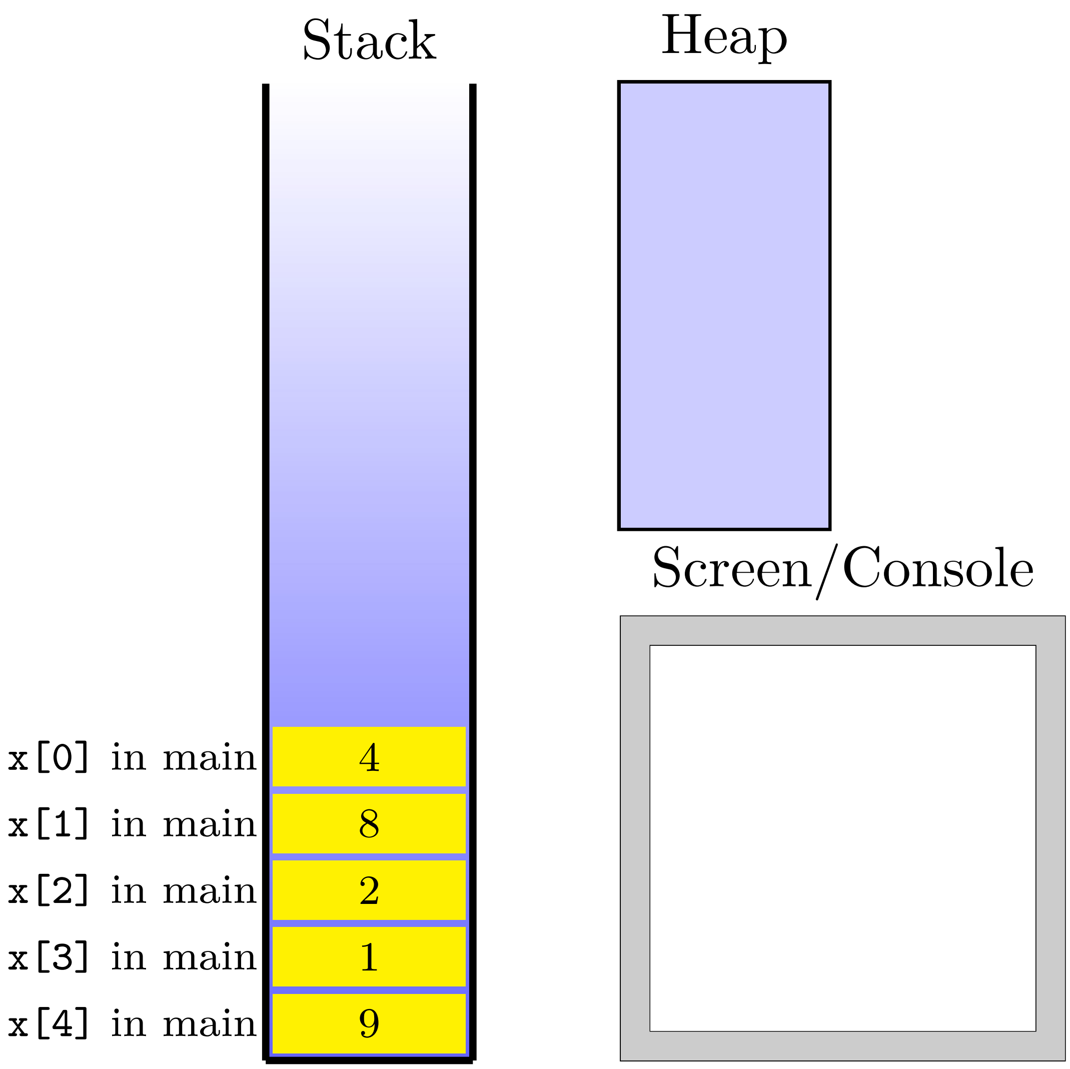
Heap

Screen/Console

Trace findMedian Program (10)

Code Segment

```
void sort(int *ptr, int len)
{            ...             }
int median(int *ptr, int len)
{
   int i, out, *p;
   p=(int *)malloc(len*
            sizeof(int));
   for (i=0; i<len; i++)
            p[i]=ptr[i];
   sort(p, len);
   out=p[len/2];
   free(p);
   return out;
}
int main()
{
   int x[5]={4, 8, 2, 1, 9};
   printf("%d\n",median(x, 5));
   return 0;
}
```



Stack

x[0] in main	4
x[1] in main	8
x[2] in main	2
x[3] in main	1
x[4] in main	9

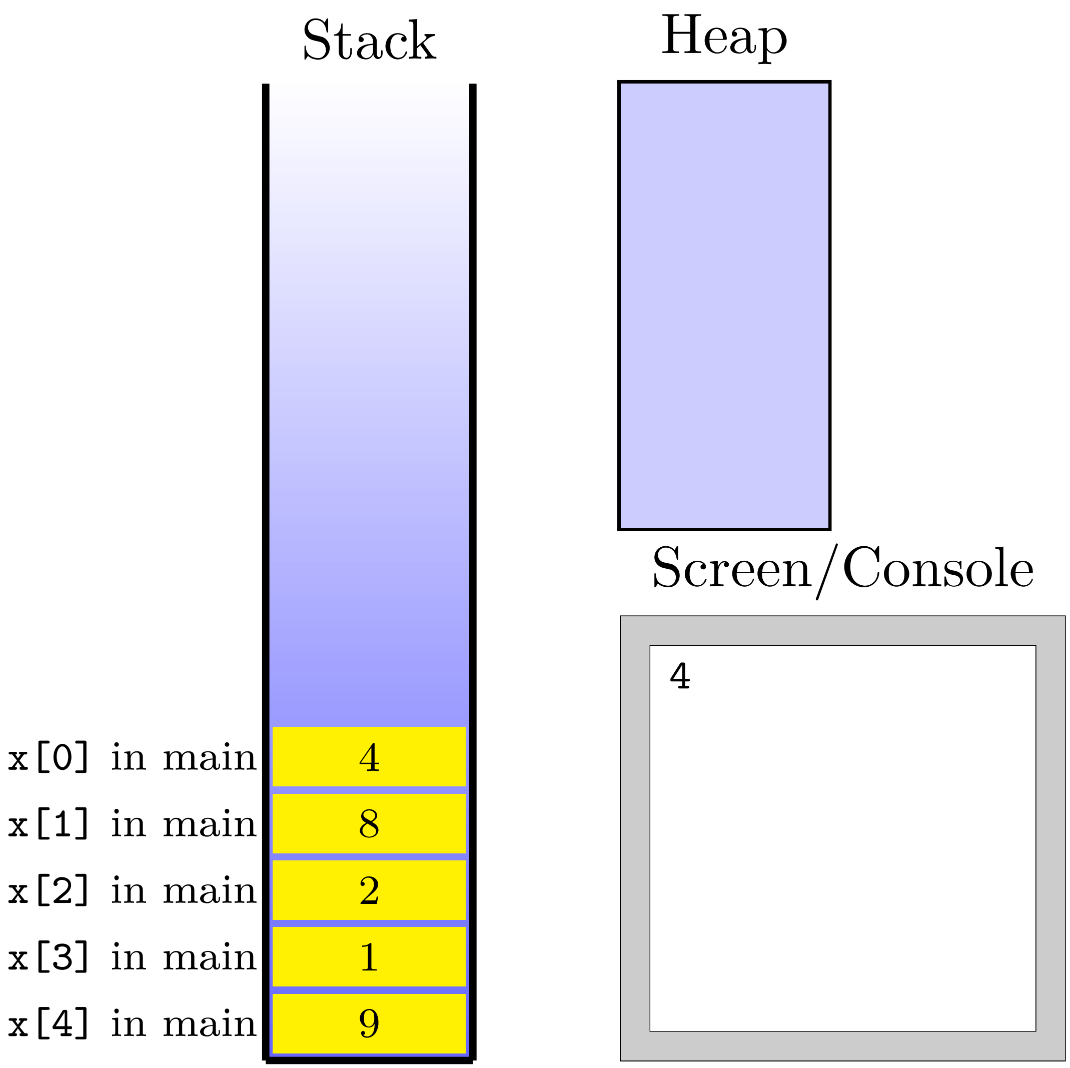
Heap

Screen/Console

Trace findMedian Program (11)

Code Segment

```
void sort(int *ptr, int len)
{            ...              }
int median(int *ptr, int len)
{
   int i, out, *p;
   p=(int *)malloc(len*
            sizeof(int));
   for (i=0; i<len; i++)
            p[i]=ptr[i];
   sort(p, len);
   out=p[len/2];
   free(p);
   return out;
}
int main()
{
   int x[5]={4, 8, 2, 1, 9};
   printf("%d\n",median(x, 5));
   return 0;
}
```



Stack

x[0] in main 4

x[1] in main 8

x[2] in main 2

x[3] in main 1

x[4] in main 9

Heap

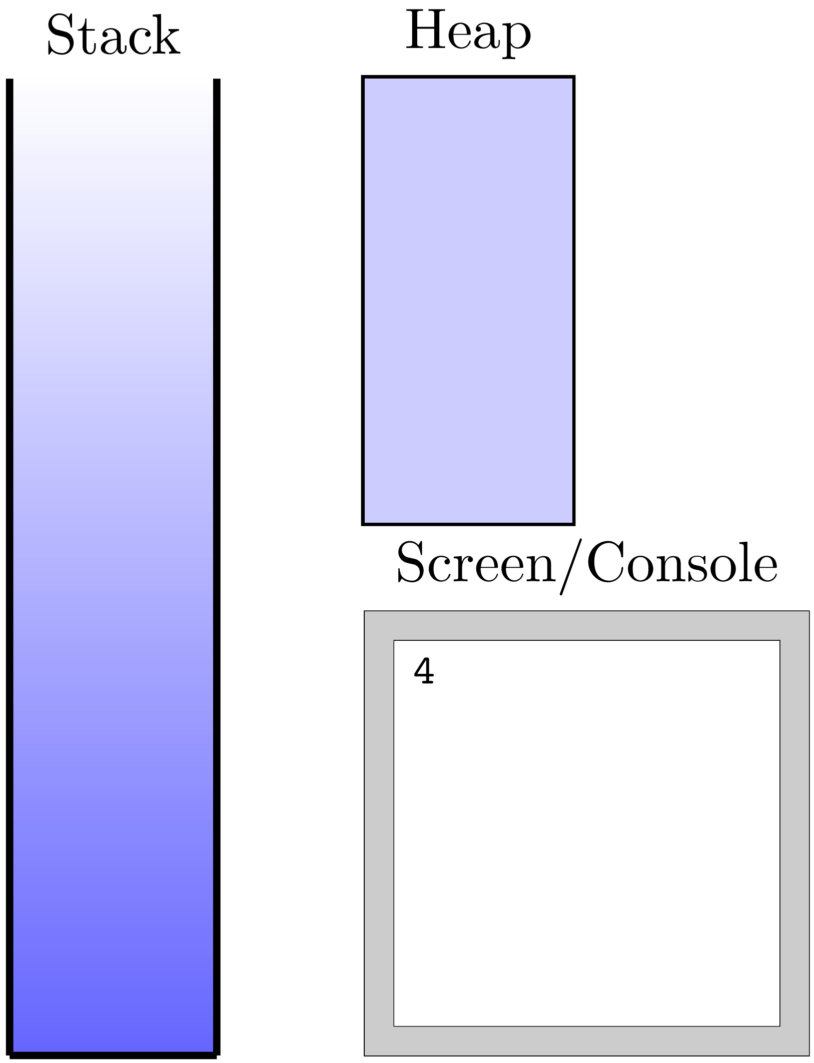
Screen/Console

```
4
```

Trace findMedian Program (12)

Code Segment

```
void sort(int *ptr, int len)
{             ...              }
int median(int *ptr, int len)
{
   int i, out, *p;
   p=(int *)malloc(len*
               sizeof(int));
   for (i=0; i<len; i++)
               p[i]=ptr[i];
   sort(p, len);
   out=p[len/2];
   free(p);
   return out;
}
int main()
{
   int x[5]={4, 8, 2, 1, 9};
   printf("%d\n",median(x, 5));
   return 0;
}
```


Stack
Heap
Screen/Console
4