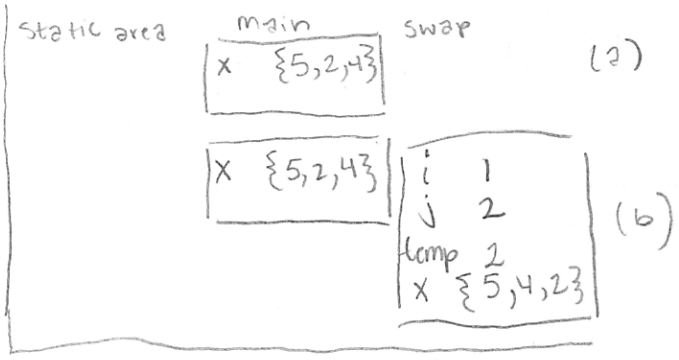
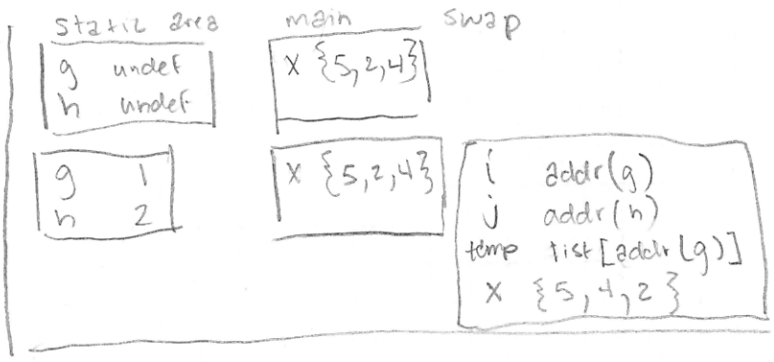
9.1
```
void swap(int[] list, int i, int j){
  int temp = list[i];
  list[i] = list[j];
  list[j] = temp;
}
void main(){
  int x[3] = {5, 2, 4};
  swap(x, 1, 2);
}
```

swap({5,2,4}, 1, 2)
int temp = list[i] = 2
x{5,_,4} → x{5,2,4} → x{5,4,4}
(i=1, j=2)

x{5,4,4} → temp → x{5,4,2}
(j=2, temp=2)




```
int g, h;
void swap(int[] list, int *i, int *j){
  int temp = list[*i];
  list[*i] = list[*j]
  list[*j] = temp;
}

void main(){
  int x[3] = {5,2,4};
  g = 1; h = 2
  swap(x, &g, &h);
```

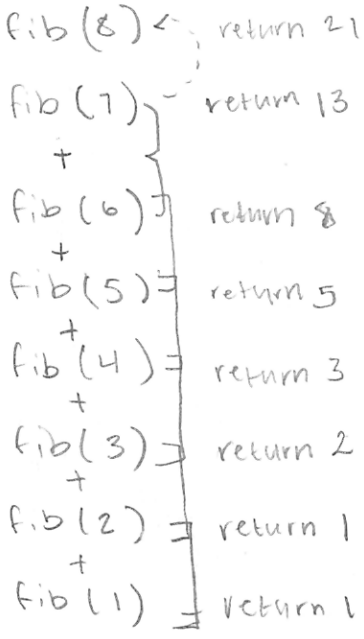


pass by value-result

```
void swap(int &a, int &b){
    int temp = a;
    a = b;
    b = temp;
void main(){
    int val = 2
    int x[3] = {5,2,4};

    int temp1 = value;
    int temp2 = x[0]

    swap(temp1, temp2);
    val = temp1;
    x[0] = temp2;

    cout<<val<<" "<<x[0]<<endl;

    temp1 = x[0];
    temp2 = x[1];
```

```
swap(x[0], x[1]);

x[0] = temp1
x[1] = temp2;

cout<<x[0]<<" "<<x[1]<<endl;

temp1 = val;
temp2 = x[val];

swap(val, x[val]);

val = temp1
x[val] = temp2

cout<<val<<" "<<x[val]<<endl;
}
```

```
int fibonacci (int n) {
  if (n<2) return n
  else return fibonacci(n-1) + fibonacci(n-2)
```



Every time Fibonacci is invoked, a new stack activation record is created. 13 stack activation records are necessary for a call to Fib(13).

A non recursive fib definition won't call "fib" in each run of fib and so won't create any more stack activation records other than the first call

```
int fib(int n){
  int fib0, fib1, temp, k;
  fib0=0; fib1=1; k=n;
  while (k>0) {
    temp = fib0;
    fib0 = fib1
    fib1 = fib0 + temp;
    k = k-1;
  }
  return fib0;
```

b.

```
int rem(int x, int y) {
  return x - x/y * y;
}
int gcd(int x, int y) {
  if (y==0) return x;
  else return gcd(y, rem(x,y));
}
int main() {
  int ans;
  ans = gcd(24,10);
```

main [ans undef]

call gcd | x 24, y 10, gcd undef
call rem | x 24, y 10, rem undef
return rem | x 24, y 10, rem 4
call gcd | x 10, y 4, gcd undef
call rem | x 10, y 4, rem undef
return rem | x 10, y 4, rem 2
return gcd | x 4, y 2, rem 0
call gcd | x 2, y 0, gcd undef
return gcd | x 2, y 0, gcd 2
return gcd | x 4, y 2, gcd 2
return gcd | x 10, y 4, gcd 2
return gcd | x 24, y 10, gcd 2
main | ans 2

7. (2) show contents of the new stack frame when the call quicksort(2) is initiated, for the array $z = \{4, 2, 5, 1\}$.

length	4
list	{4,2,5,1}

frame for quicksort

j	4
i	1
key	4
n	4
m	1

frame for sort

temp	4
j	1
i	4

frame for swap

... until we derive $z = \{1, 2, 4, 5\}$