1.

1) In pass by value, a copy of variable's value passed to function as a parameter. whereas in pass by reference the memory address of variable (pointer) is passed to function Also in pass by value changes to parameter doesn't affect original variable, whereas in pass by reference changes made to variable via dereferencing the pointer affects original variable.

2) variable x is passed by value. Its value didn't change after the function call as the function only received a copy of x. Whereas, variable y is passed by reference as its address is passed to the function via & operator. also, its value is changed after The function call, as the pointer holding its memory address was dereferenced.

2.

error 1: missing #include <stdio>
need This for printf.

```
int main() {

    int A[5]={1,3,7,4,0};

    int *P[5], *pA;

    int i, x, y;

 //Set pointer to the base address of array
 *pA = &A;
```

error 2: should be pA = &A[0]; or pA = A;
not *pA because * dereferences ptr.
and A acts as a ptr, can't use &A.

```
 // Assign A[1] to x and y
 x = *pA+1;
```

error 3: wrong ptr arithmetic.
it adds 1 to A[0] instead of access A[1]
should be x = *(pA+1);

```
 A++;
```

error 4: INCORRECT, can't add 1 to an array.
remove line.

```
 y=*A;
```

errors 5: should be y = *(A+1)

```
 printf("x = \t%d",x);
 printf("y = \t%d",y);


    //Set every pointer to one array element

    for(i=0;i<5;i++)

      *P[i]=&A[i];
```

erro

error 6: should be P[i] = &A[i];
b/c we can't dereference P[i] before
we asign it.

```
    //print array elements using pointers

    for(i=0;i<5;i++)

      printf("\t%d",P[i]);
```

error 7: This prints the address not
the element.
should be: printf("\t%d", *P[i]);

```
    return 0;

 }
```

final code on next page (w/ corrections.

```
#include <stdio.h>
 int main () {
   int A[5] = {1,3,7,4,0};
   int *P[5], *pA;
   int i, x, y;
   pA = A;
   x = *(pA +1);
   y = *(A +1);
   printf ("x = \t%d\n", x);
   printf ("y = \t%d\n", y);

     for (i = 0; i<5 ; i++)
        p[i] = &A[i];

      for (i=0 ;i <5; i++)
         printf ("\t %d \n", *P[i]);
      return 0;
```

c doesn't care about indents. also don't need {} for 1 line of code, just blocks.

3.

1) show $3n^2 = O(n^3)$ use $f(n) \le \Theta(g(n))$

$3n^2 \le c \cdot n^3 \quad \forall n \geq n_0$

$3 \le c \cdot n$

$c = 3 \quad n_0 = 1$

inequality holds for $c = 3$ and $n_0 = 1$ $\forall n \geq 1$.

2)

$n^2 + 2n \neq \Omega(n^3)$

$f(n) \geq c \cdot g(n) \quad \forall \; n \geq n_0$

$n^2 + 2n \geq c \cdot n^3$

$\frac{n^2}{n^3} + \frac{2n}{n^3} \geq c$

$\frac{1}{n} + \frac{2}{n^2} \geq c$ now see as $n \to \infty$

$\lim_{n \to \infty} \cancelto{0}{\frac{1}{n}} + \cancelto{0}{\frac{2}{n^2}} = 0$

so $0 \geq c$ but $c \in R^+$ contradiction

$\therefore n^2 + 2n \neq \Omega(n^3)$.

3) $n^2 - 2n = \Theta(n^2)$

$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \quad \forall \; n \geq n_0$

find c_1, c_2.

$c_1 \cdot n^2 \leq n^2 - 2n \leq c_2 \cdot n^2$

$c_1 n^2 \leq n^2 - 2n$ $\qquad$ $n^2 - 2n \leq c_2 n^2$

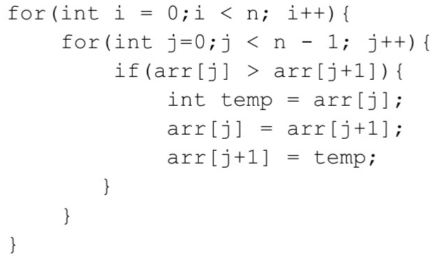
$c_1 = 1/2$ $\qquad$ $c_2 = 1$

$n^2 - 2n = \Omega(n^2)$ $\qquad$ $n^2 - 2n = O(n^2)$

$\therefore n^2 - 2n = \Theta(n^2)$.

4. regular bubble sort

loop runs n times
inner loop n-1 times.



```
for(int i = 0;i < n; i++){
    for(int j=0;j < n - 1; j++){
        if(arr[j] > arr[j+1]){
            int temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
        }
    }
}
```

} 3. operations in swap.

Best case: $x = [1, 2, 3, \dots n]$

$n(n-1)$ comparisons

no swaps.

$n \cdot (n-1)$ operations

Worst case: $[n, n-1, n-2, \dots 1]$

$n(n-1)$ comparisons

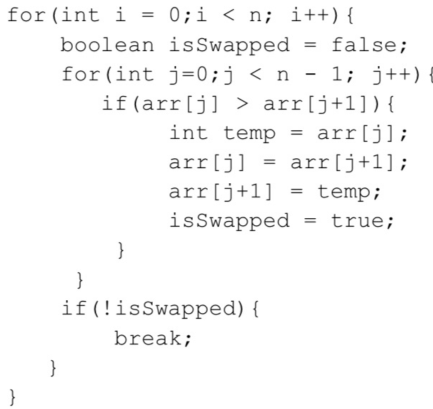
$n(n-1) \times 3$ for swap operations

$$n(n-1) + 3n(n-1) = n^2 - n + 3n^2 - 3n$$
$$= 4n^2 - 4n = 4n(n-1)$$

$\therefore$ $4n(n-1)$ total operations

revised bubble sort

runs atleast once.



```
for(int i = 0;i < n; i++){
    boolean isSwapped = false;
    for(int j=0;j < n - 1; j++){
       if(arr[j] > arr[j+1]){
            int temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
            isSwapped = true;
        }
     }
    if(!isSwapped){
        break;
    }
}
```

} 4

Best case:

outer loop runs once (no swaps it breaks).

is swapped = false.

inner loop runs n-1 times

since array is sorted no swap ops occur, but n-1 comparisons.

if (!isswapped) one operation.

$1 + (n-1) + 1 = \underline{n+1}$ operations

Worst case

outer loop: runs n times

inner loop: n-1, n-2.....1 $= n\frac{(n-1)}{2} \approx n^2$

4 operations after. $4 \times n\frac{(n-1)}{2}$

$n\frac{(n-1)}{2} + 3 \times n\frac{(n-1)}{2} + n$

$= 2n^2 - n$

5.

old computer: $T(n) = k \cdot n^2$

$$T(100) = k \cdot 100^2 = 1 \text{ minute}$$

$$k \cdot 100^2 = 1$$

$$k = 1/100^2 = 1/10000$$

$$T(n) = n^2/10000$$

time for new computer:

$$T(n) = \frac{T_{old}(n)}{64}$$

$$= \frac{n^2}{10000} \times \frac{1}{64} = \frac{n^2}{640000} = 1 \text{min}$$

$$n_{new} = \sqrt{640000} = 800$$

n = 800 in 1 minute.