Assignment 2
Daniel Grew
0978547

Question 1.

Algorithm BFInversions (A[0.. n-1])
// Counts number of inversions in an array
// Input : A[0.. n-1] array of integers
// Output : number of Inversions

```
count ← ∅
for i ← ∅ to n-2
    for j ← ∅ to n-1
        if A[i] > A[j]
            count ← count + 1
return count
```

Analysis: Basic Operation : Comparison $A[i] > A[j]$

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-1} (1) = \sum_{i=0}^{n-2} (n-1-i)$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} (i)$$

$$= n^2 - 2n + 1 - \frac{1}{2}(n^2 - 3n + 2) \in \underline{\Theta(n^2)}$$

Algorithm MergeSortInversions (A[0.. n-1])
// Counts number of inversions recursively
// Input : A[0.. n-1] Array of integers
// Output : number of Inversions

```
if n == 1
    return
count ← ∅
Copy A[0.. ⌊n/2⌋-1] to B[0.. ⌊n/2⌋ -1]
Copy A[⌊n/2⌋.. n-⌊n/2⌋ -1] to C[0.. n-⌊n/2⌋ - 1]
x ← MergeSortInversions (B)
y ← MergeSortInversions (C)
return MergeInversions (B, C, A) + x + y
```


Scanned with CamScanner

```
Algorithm MergeInversions(B[0..p-1], C[0..q-1], A[0..p+q-1])
  // Calculates the number of inversions between two sorted sets
  // Input: B[0..p] and C[0..q]: both sorted arrays of integers
  // Output: number of inversions
  i ← 0; j ← 0; k ← 0; inversions ← 0
  while i < p and j < q do
        if B[i] ≤ C[j]
              A[k] ← B[i]
              if i < p-1
                    inversions ← inversions + j
              i ← i+1
        else  A[k] ← C[j]
              if B[i] ≠ C[j]
                    inversions ← inversions + 1
              j ← j+1
        k ← k+1
  if i = k
      Copy C[j..q-1] to A[k..p+q-1]
  else Copy B[i..p-1] to A[k..p+q-1]
      for index ← i to p-1 do
            inversions ← inversions + j
  return inversions
```


Scanned with CamScanner

Analysis:

$C(n) = 2C(n/2) + C_{merge}(n)$

$C_{best} = 2C(n/2) + n/2$

$a = 2,\ b = 2,\ d = 1 \leftarrow f(n) = n/2 \in \Theta(n)$

So, $C_{best}(n) \in \Theta(n \log(n))$

$C_{worst}(n) = 2C(n/2) + n-1 \quad : \quad f(n) = n-1 \in \Theta(n)$

$a = 2,\ b = 2,\ d = 1$

So, $a = 2^1 = b^d$ and

$C_{worst}(n) \in \Theta(n \log(n))$

Initial Condition for Best Case and Worst Case:

$C_{best}(1) = C_{worst}(1) = \emptyset$

Comparison: Theory:

Brute force: $C(50{,}000) = 2.5 \cdot 10^9$

Θ MergeSort Based: $C(50{,}000) = 234{,}949$

Question 2.

```
Algorithm BFConvexHull(P[0..n-1])
  // Finds all points that make up the convex hull
  // Input: Array of Points P[0..n-1]
  // Output: Array of Points that make up the convex hull
  i← ∅; j← ∅; k← ∅; side← ∅; hull← ∅;
  for i← ∅ to n-1 do
     for j← ∅ to n-1 do
        c← (P[i].x * P[j].y) - (P[i].y * P[j].x)
        a← P[j].y - P[i].y
        b← P[i].x - P[j].x
        for k←0 to n-1
             q← a* P[k].x + b* P[k].y
             if q > c ~~and sign of side same~~
                 if side change
                      break
             else if side change
                 break
        if k=n
             ~~hull~~ ~~copy~~ Hull[hull] ← P[k]
             hull ← hull+1
             break
  return Hull[0.. hull-1]
```


Scanned with CamScanner

Analysis: Basic Operation: Comparison : if $q > c$

$$C(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} (1)$$

$$= \sum_{i=0}^{n-1} \sum_{i=0}^{n-1} (n) = n(n)(n) = n^3 \in \Theta(n^3)$$

Algorithm DivideConquerConvexHull(P[0..n-1])

```
// Finds all points that make up the convex hull
// Input : Array of Points P[0..n-1]
// Output : Array of Points on Convex Hull
q; r;
for i ← 0 to n-1
    if P[i].x < q.x
        q ← P[i]
    else if P[i].x > r
        r ← P[i]
U ← SplitHull (q, r, upper)
L ← SplitHull (q, r, lower)
UpperHull ← QuickHull (U[0..p], q, r)
LowerHull ← QuickHull (L[0..q], q, r)
ConvexHull [0] ← q
Copy UpperHull[0..m-1] to ConvexHull[1..m]
ConvexHull[m+1] ← r
Copy LowerHull[0..n] to ConvexHull[m+2+n-1]
return ConvexHull
```


Scanned with CamScanner

```
Algorithm QuickHull (P[0.. n-1], p1, p2, hull)
    // Recursively finds the points that make up the convex hull
    // Input: Array of Points P[0.. n-1], and two points that are on the convex hull
    // Output: Array of Points that make up the specified side of the hull
    if n = 0
        return []
    farthest ← P[0]
    for i←0 to n-1 do
            if P[i] is the farthest point
                    farthest ← P[i]
    Hull1points ← SplitHull (P[0.. n-1], p1 farthest )
    Hull2points ← SplitHull ( P[0.. n-1], p2, farthest)

    Hull1 ← QuickHull (Hull1points[0.. p], p1, farthest, hull)
    Hull2 ← QuickHull (Hull2points [0.. q], p2, farthest, hull)

    Hull [0.. m+n];
    Copy Hull1[0.. m-1] to Hull [0.. m-1]
    Hull[m] ← farthest
    Copy Hull2 [0.. n] to Hull [m+1 .. m+n]
    return Hull [0.. m+n]
```


Scanned with CamScanner

Analysis: Initial Condition for Best Case and Worst Case: $C_{best}(0) = C_{worst}(0) = \phi$

Best and worst case for SplitHull is $C_{SH}(n) = n-1 \in \Theta(n)$

Best and worst case for finding furthest point $C(n) = n-1 \in \Theta(n)$

In the best case scenario, the points are not all along the convex hull. This way more points can be eliminated from consideration. Moreover, in the best case, there will be an equal number of points on both sides of the split $(n/2)$

$C_{best} = 2C_{best}(n/2) + n+1 + n+1 = 2C_{best}(n/2) + 2n+1$

Here, $f(n) = 2n+1 \in \Theta(n)$. So, $a=2$, $b=2$, $d=1$

Hence, According to the master theorem: $C_{best}(n) \in \Theta(n\log(n))$

The worst case, similar to QuickSort, happens when SplitHull splits the Array into in such a way that most points fall into one of the Arrays.

$C_{worst}(n) \in \Theta(n^2)$

In the best case, we have that the QuickHull Algorithm is much faster $(n\log(n))$ than the brute force Algorithm (n^3).

We also see this in the time the implementation takes to complete.

The QuickHull Algorithm is also faster in the worst case. (n^2)

~~Theory:~~

Theory: QuickHull

Best : $C(30{,}000) = 30{,}000 \log(30{,}000)$
$= 134{,}313$

Worst : $900{,}000{,}000$

Theory Brute force : $2.7 \cdot 10^{13}$

Scanned with CamScanner