Assignment 2 – Algorithm Design

1.1

```
ALGORITHM BruteForceInversions(A[0...n-1])
  // A is an array of n distinct numbers
  // Returns the number of inversions in the array
  inversions = 0
  for i = 0 to n - 1 do
    for j = i + 1 to n - 1 do
      if A[i] > A[j] then
        inversions = inversions + 1
  return inversions
```

Basic operation: Comparison

Repetitions: $(n-1)+(n-2)+\cdots+2+1=\frac{n(n-1)}{2}\in\Theta(n^2)$

1.2

```
ALGORITHM SortAndCountInversions(A[0...n-1], left, right)
  // A is an array of n distinct numbers
  // Returns the number of inversions in the array

  // Base case: Array of length 0 or 1
  if left >= right + 1 then
    return 0
  mid = ⌊(left + right) / 2⌋

  // Count inversions and sort recursively
  inversions = SortAndCountInversions(A, left, mid) +
               SortAndCountInversions(A, mid, right)

  // Linear merge of left and right sorted lists
  initialize list S
  i = left
  j = mid
  while i < mid and j < right do
    if A[i] <= A[j] then
      add A[i] to S
      i = i + 1
    else
      add A[j] to S
      // Whenever an element crosses from the right list into
      // the left list, compute how many elements it was greater than
      // in the left list and add it to the total
      inversions = inversions + (mid - i)
      j = j + 1
  while i < mid do
    add A[i] to S
    i = i + 1
  while j < mid do
    add A[j] to S
    j = j + 1

  // Copy over sorted elements to A
  for k = left to right - 1 do
    A[k] = S[k - left]
```

Basic operation: Comparison

Repetitions: (best case)

$$
\begin{aligned}
T(n) &= 2T(\frac{n}{2}) + \frac{n}{2} \text{ and } T(1) = 0 \\
&= 2T(2^{k-1}) + 2^{k-1} && (n = 2^k) \\
&= 2(2T(2^{k-2}) + 2^{k-2}) + 2^{k-1} \\
&= 2^i T(2^{k-i}) + i2^{k-1} \\
&= 2^k T(2^{k-k}) + k2^{k-1} && (\text{sub } i = k) \\
&= 2^k T(1) + \frac{k2^k}{2} \\
&= 0 + \frac{k2^k}{2} \\
&= \frac{n\log_2 n}{2} && (\text{sub } 2^k = n) \\
&\in \Theta(n\log n)
\end{aligned}
$$

Master Theorem:

$$
\begin{gathered}
T(n) = 2T(\frac{n}{2}) + \frac{n}{2} \text{ and } T(1) = 0 \\
f(n) = n/2 \in \Theta(n^1) \\
a = 2, b = 2, d = 1 \\
b^d = 2^1 = 2 = a
\end{gathered}
$$

Thus by the Master Theorem, $T(n) \in \Theta(n^1 \log n)$

1.3

	Brute Force	Divide and Conquer
Best case	$\Theta(n^2)$	$\Theta(n\log n)$
Average case	$\Theta(n^2)$	$\Theta(n\log n)$
Worst case	$\Theta(n^2)$	$\Theta(n\log n)$
Runtime	266 ms	10 ms

2.1

```
ALGORITHM ShortestPath(P[0...n - 1], s1, s2)
  // P is an ordered list of points
  // s1 and s2 are indices of two points in P
  // Returns the points in the shortest path from s1 to s2 in P
  // and the length of the path

  initialize list L
  leftLength = 0
  lastPoint = s1
  for i = 0 to n - 1 do
    j = (s1 - i) mod n
    p = P[j]
    leftLength = leftLength + distance from lastPoint to p
    lastPoint = p
    add p to L
    if j == s2 then
      break

  initialize list R
  rightLength = 0
  lastPoint = s1
  for i = 0 to n - 1 do
    j = (s1 + i) mod n
    p = P[j]
    rightLength = rightLength + distance from lastPoint to p
    lastPoint = p
    add p to L
    if j == s2 then
      break

  if leftLength < rightLength then
    return L, leftLength
  else
    return R, rightLength

ALGORITHM HullSortClockwise(H[0...n - 1])
  // H is a set of points in a convex hull
  // Returns a clockwise-ordered list of points from H
  if n == 0 then
    return

  lowest = H[0]
  for i = 1 to n - 1 do
    if H[i].y < lowest.y then
      lowest = H[i]
```

```
  initialize list A[0...n - 1]

  for i = 0 to n - 1 do
    A[i] = atan2(H[i].y - lowest.y, H[i].x - lowest.x)

  Sort H using values in A

ALGORITHM BruteForceConvexHull(S[0...n - 1])
  // S is a set of points in a 2-dimensional plane
  // Returns a clockwise-ordered list of points in the convex
  // hull of S

  initialize list H

  for i = 0 to n - 1 do
    for j = 0 to n - 1 do
      A = S[i]
      B = S[j]

      a = B.y - A.y
      b = A.x - B.x
      c = A.x * B.y - A.y * B.x

      isHullSegment = False
      side = undecided

      for k = 0 to n - 1 do
        C = S[k]

        if (a * C.x + b * C.y) > c then
          if side == left then
            isHullSegment = False
            break
          side = right
        else if (a * C.x + b * C.y) < c then
          if side == right then
            isHullSegment = False
            break
          side = left
        else
          if min(A.x, B.x) <= C.x <= max(A.x, B.x) and
             min(A.y, B.y) <= C.y <= max(A.y, B.y) then
            continue
          isHullSegment = False
          break

      if isHullSegment then
```

```
        add A to H

    // Removes duplicate points in H
    RemoveDuplicates(H)

    // Sorts points in P in a clockwise orientation
    HullSortClockwise(H)

    return H
```

The algorithm to solve the shortest path around problem consists of two steps:

- Finding the points in the convex hull of S
- Traversing the convex hull to determine the shortest path from s1 to s2

Brute Force Convex Hull

Basic operation: Comparison

Repetitions:

$$\begin{aligned}&n^3 + n^2 \text{ (remove duplicates)} + n \log n \text{ (clockwise sort)} \\ &\in \Theta(n^3)\end{aligned}$$

Shortest Path

Basic operation: Addition

Repetitions: $n \in \Theta(n)$

2.2

```
ALGORITHM Partition(S[0...n - 1], A, B)
  // Partitions S in-place into S1: points that are to the left or on the line
  // and S2: points that are strictly to the right of the line
  // Returns the start index of S2
  if n == 0 then
    return
  a = B.y - A.y
  b = A.x - B.x
  c = A.x * B.y - A.y * B.x

  f = left
  for i = 0 to right - 1 do
    if (a * S[i].x + b * S[i].y) <= c then
      swap S[i] and S[f]
      f = f + 1

  return f

ALGORITHM FindHull(H[0...m - 1], S[0...n - 1], A, B)
  if n == 0 then
    return
  C = left
  for i = left to right - 1 do
    if S[i] is farther from line AB than C then
      C = S[i]

  add C to H
  remove C from S

  // Divide S into S1: points to the right of the line AC
  //           and S2: points to the right of the line CB
  p = partition(S, A, B)
  q = partition(S[0...p], C, B)

  FindHull(H, S[p...n - 1], A, C)
  FindHull(H, S[q...p], C, B)


ALGORITHM DivideAndConquerConvexHull(S[0...n - 1])
  // S is a set of points in a 2-dimensional plane
  // Returns a clockwise-ordered list of points in the convex
  // hull of S
  initialize list H

  min = 0
  max = 0
```

```
for i = 1 to n - 1 do
  if S[i].x < S[min].x or (S[i].x == S[min].y and S[i].y < S[min].y) then
    min = i
  if S[i].x > S[max].x or (S[i].x == S[max].y and S[i].y > S[max].y) then
    max = i

A = S[min]
B = S[max]

add A and B to H
remove A and B from S

// Divide S into S1: points to the left of the line
//           and S2: points to the right of the line
p = Partition(S, A, B)
q = Partition(S[0...p], B, A)

FindHull(H, S[p...n - 1], A, B)
FindHull(H, S[q...p], B, A)

// Sorts points in P in a clockwise orientation
HullSortClockwise(H)

return H
```

Divide and Conquer Convex Hull

Basic operation: Comparison

Repetitions: (best case)

$$
\begin{aligned}
T(n) &= 2T(\frac{n}{2}) + n \text{ and } T(1) = 0 \\
&= 2T(2^{k-1}) + 2^k && (n = 2^k) \\
&= 2(2T(2^{k-2}) + 2^{k-1}) + 2^k \\
&= 2^i T(2^{k-i}) + i2^k \\
&= 2^k T(2^{k-k}) + k2^k && (\text{sub } i = k) \\
&= 2^k T(1) + k2^k \\
&= 0 + k2^k \\
&= n \log_2 n && (\text{sub } 2^k = n) \\
&\in \Theta(n \log n)
\end{aligned}
$$

Master Theorem:

$$
\begin{gathered}
T(n) = 2T(\frac{n}{2}) + n \text{ and } T(1) = 0 \\
f(n) = n \in \Theta(n^1) \\
a = 2, b = 2, d = 1 \\
b^d = 2^1 = 2 = a
\end{gathered}
$$

Thus by the Master Theorem, $T(n) \in \Theta(n^1 \log n)$

Shortest Path

Same as 2.1

	Brute Force	Divide and Conquer
Best case	$\Theta(n^3)$	$\Theta(n \log n)$
Average case	$\Theta(n^3)$	$\Theta(n \log n)$
Worst case	$\Theta(n^3)$	$\Theta(n^2)$
Runtime	14581 ms	1 ms