PART A

Hedcuter summary

1. Computing Voronoi diagram-

Algorithm

1. Construct a virtual image (res) from the input image(img) but changing the resolution as per the number of sub-pixels set by the user.
   res.dimensions = subpixels*img.dimensions
1. Initialize dist, root and visited arrays with each index storing max of float, max of unsigned short, and 0 respectively.

dist=> an array of size res.height*res.width , that is it covers all points in **res.** dist stores distance values for each point in res, where in the value d corresponds to the intensity of color at that point(d = 256 - intensity(x,y)

root=>an array of size same as dist and stores all voronoi site ids.

visited=> an array of size same as dist and stores 1 if a point is visited else 0

1. For each voronoi cell, take hold of the cell_site and set values for dist and root at the location of the cell_site, also store a pair(dist.value and root.id) for each cell_site in an array **open**
2. Create a heap from the array **open** to get a partially sorted structure on the basis of dist.value for each pair in **open**
3. For each pair in heap
   1. check if dist.value is updated
   2. visited is 1
   3. Consider a 3x3 grid with current cell in the center
      For each neighbor of current cell,
      a. calculate newd(newd
      = currentCells d value + 256 - intensity(x,y)
      where (x, y) are neighbor cells positions.
      b. compare with old = dist.value at (x,y)
      c. if newd< oldd
      store dist.value for this neighbor cell
      set root at the position of this neighbor cell as the root at current cell (that is they belong to same voronoi site)
      insert this neighbor cell in the heap

1. For all positions of the res image, add positions with same root to the coverage of the cells with that root id.
2. remove cells with empty coverage

Computing Centroidal Voronoi Tesselation-

Algorithm:

1. For max number of iterations or until the sites are not moving towards the cell centroid
   do
   1. calculate voronoi of img
   2. calculate distance to move the sites (offset)
   For each cell
   a. find a weighted average of colors of cells in the coverage of current cell.
   Normalize this weighted average
   calculate distance between current cell site and a position with the weighted average as its x and y. (this is a centroid for current cell)
   update current cell's site with this new x,y , that is with the centroid.
   3. we can either take an average of all cell's offset or just the largest of all the offsets, as the distance to move our sites.

Stippling method-

Algorithm:

1. For each cell from Centroidal Voronoi Tessellation
   1. For each cell in the coverage of current cell
      calculate average color intensities of gray, red, green, blue
   2. create a disk centered at this site, and color vector as calculated in step 1.

Voronoi summary

1. Computing Voronoi diagram-

The code makes use of Fortune's algorithm to create voronoi diagrams.
Algorithm

1. n = number of stipple points to generate
2. generate uniform random n points
   i. from 0 to ImageWidth for x co-ordinates of the points
   ii. from 0 to ImageHeight for y co-ordinates of the points
1. These n points obtained are treated as the voronoi sites for which Fortune's algorithm is to be applied to create a voronoi diagram
2. PQ = a priority queue to store events (sites encountered as we sweep so that we sweep from y-min to y-max and if the sites have same y, from x-min to x-max)
3. Also use x-min, y-min and x-max, y-max for bounding box
4. EL = a doubly linked list to store half edges between the arcs traced by the sweep line and sites

while PQ is not empty

1. When a site event occurs, that is on picking next point (starting from lowest in PQ), 2 new edges are to be inserted in EL.
   This point creates a new arc on the beach line, which traces 2 additional edges.

For left arc - ELpm 0

i. First obtain half edge to the left of the event (site)
ii. create a bisector between the new event (site) and the bottom site.
iii. add this bisector to the right of left half edge obtained in 7.i
iv. if the left edge and this bisector intersect, get the vertex of the left edge and add it as vertex for the bisector.

Remove left edge from PQ and add this bisector in its place as the left edge is not being traced any more by the sweep line.

We need to do the same for the second additional edge

For right arc - ELpm 1

i. We have half edge to the left of this arc, which is the bisector added in earlier step
ii. add this bisector to the right of the previous bisector in EL
iii. obtain half edge to the right of the event (site)
iii. if the right half edge and this bisector intersect, get the vertex of the right half edge and add it as vertex for this bisector.
iv. add this bisector to PQ

1. To handle a circle event, we need to get a vertex from the left of current site and one from the right. This can be done using EL.
   i. eleft = edge to the left of current site (event)
   bottomRegion = region bounded by eleft

ii. leleft = the edge to the left of eleft in EL.
The vertex of leleft is one of our points.
iii. eright = edge to the right of current site (event)
topRegion = region bounded by eright
iv. reright = the edge to the right of eright in EL.
The vertex of reright is another of our points.
v. These two points together with current site (event) form a circle.
vi. Make this new site as a vertex and set end points for eleft and eright as this vertex.
vii. Mark eright and eleft for deletion from EL
viii. Swap topRegion and bottomRegion if bottomRegion is higher than topRegion.
ix. We now need to connect leleft and reright to the bisector of topRegion and bottomRegion. We do this as we did in step 7.

end while

1. Connect all half edges with source and end vertices. Store an edgeMap with site and edge as a pair for each edge.

Computing Centroidal Voronoi Tesselation-

Algorithm:

We are calculating the centroid for each cell (site and edges surrounding this site, which we stored in edgeMap while creating voronoi diagram).

Using edgeMap
1. For each site pSite
For each bounding edge siteEdge
Generate a clipping line lClip such that pSite is on the same side of half plane formed by lClip.

For each site
1. get the dimensions of the site by finding
i. extent of each site. (maxX, maxY and minX, minY)
ii. cellWidth = (maxX-minX) x numberOfSubPixels
cellHeight = (maxY-minY) x numberOfSubPixels
1. let xsteps = cellWidth
ysteps = cellHeight

```
xstep = 0
ystep = 0
potentialCentroid = minX, minY
        for ysteps number of times
                ystep++
                for xsteps number of times
                        xstep++
                        potentialCentroid = (minX+xstep, minY+ystep)
                        for each half planes lClip from 1
                        check if potentialCentroid lies on the same
        side
```

Stippling method-

Algorithm:

For each site

1. get the dimensions of the site by finding
   i. extent of each site. (maxX, maxY and minX, minY)
   ii. cellWidth = (maxX-minX) x numberOfSubPixels
   cellHeight = (maxY-minY) x numberOfSubPixels
1. let xsteps = cellWidth
   ysteps = cellHeight

```
xstep = 0
ystep = 0
centroidRegionDensity = 0
maxCentroidRegionDensity = 0
xNormalizeSum = 0
yNormalizeSum
potentialCentroid = minX, minY
        for ysteps number of times
                ystep++
                for xsteps number of times
                        xstep++
                        potentialCentroid = (minX+xstep, minY+ystep)
                        for each half planes lClip from 1
                        check if potentialCentroid lies on the same
        side
                        if potentialCentroid lies inside for all half-lines
                                centroidRegionDensity += image intensity at potentialCentroid
```

maxCentroidRegionDensity += white color (255f)
xNormalizeSum += image intensity at potentialCentroid * potentialCentroid.x
yNormalizeSum += image intensity at potentialCentroid * potentialCentroid.y

1. pointToDrawStipple.x = xNormalizeSum/centroidRegionDensity

   pointToDrawStipple.y = yNormalizeSum/centroidRegionDensity

1. radiusForStipple = find closest (if no overlap allowed, shortest distance) or farthest (if overlap allowed, largest distance) of distances between pointToDrawStipple and edges in the site.

PART B

Image considered for these test runs- hedcuter/images/phoenix.png

Parameter	Voronoi	Hedcuter
1. Rerunning results	Output images are same Time to make images changes **Run1**: Completed in 3.06 seconds. **Run2**: Completed in 3.28 seconds.	Output images are not same Time to make images changes **Run1**: Total time: 80.8803 **Run2**: Total time: 82.2646
2. Varying the number of disks	Produces different results. This is because the minimum distance between sites reduces on increasing the number of disks. It causes some half planes to be ignored as the distance is too small. **Time**: Completed faster Completed in 2.97 seconds.	Produces different results. This is because there are more points to sample. **Time:** Took more time Total time: 99.0689

3. Size of image	Produces different results as we use uniform distribution to generate initial distribution.	Produces different results as we use uniform and Gaussian distribution to generate initial distribution.
4. Image brightness/contrast	Produces different results as we multiply the pixel intensity with its location before normalizing it with the white (highest) value for each pixel.	Produces different results as we take an average of color intensities for pixels in the coverage of a site.
5. Type of image	Produced better results than hedcuter for artificial/machine photo.	Produced better results than Voronoi for natural photos. Image considered for this test run- hedcuter/images/squirrel.png Time: Total time: 35.0567
6. Comparison with hedcuts from Wall Street Journal (WSJ)	They were different and were not as good as WSJ.	They were different and were not as good as WSJ.

Part C

Improvements in hedcuter:

i. Size of the disks

Originally the size of a disk (for a cell) was calculated as follows:

1. avgGrayScale = average grayScale value over all points of the image, which lie in the coverage of the cell
2. diskSize = 100 * DISK_SIZE / (avgGrayScale + 100)

where DISK_SIZE is by default 1 or is supplied as a parameter to main (as radius).

Improvement:
While adding points to the coverage of a cell, I am storing closest and the farthest point distances with respect to the site co-ordinates.
Then like in voronoi, I am making use of these distances to set the radius of the disk.

Executing for closesDistance had little or no effect on clustering

ii. Color of disks

Originally the color of the disk (for a cell) was calculated as follows:

1. for a cell

r = sum of red values at each point in the coverage of the cell
g = sum of green values at each point in the coverage of the cell
b = sum of blue values at each point in the coverage of the cell

then normalize each like
r = r/(number of points in the coverage of this cell)
g = g/(number of points in the coverage of this cell)
b = b/(number of points in the coverage of this cell)

These values for r, g and b determine the color of the disk

Improvement:
While adding points to the coverage of a cell, I plan to store the r, g and b values as
for each point p in the coverage of the cell
rxsum += p.x * (r at x,y)
rysum += p.y * (r at x,y)
rAreaSum = sum of r values for each point in the cell coverage

gxsum += p.x * (g at x,y)
gysum += p.y * (g at x,y)
gAreaSum = sum of g values for each point in the cell coverage

bxsum += p.x * (b at x,y)
bysum += p.y * (b at x,y)
bAreaSum = sum of b values for each point in the cell coverage

And then normalizing these values

r = (rxSum+rySum)/rAreaSum
g = (gxSum+gySum)/gAreaSum
b = (bxSum+bySum)/bAreaSum

Side notes:

Tried running hedcuter with subpixel 5 to compare voronoi output for sub pixel 5. However, the code took too much time. I terminated it after 1.25 hrs.