1. (a) Describe an efficient $(1-1/k)$-approximation algorithm for the maximum k-cut problem.

Solution: Consider the vertices one at a time in arbitrary order, and greedily assign each vertex to the set S_i that maximizes the partition cost so far. To make the algorithm (and analysis) concrete, arbitrary index the vertices $v_1, v_2, \dots, v_n$.

```
ApproxMaxKCut(G, w, k):
  for j ← 1 to k
      S_j ← ∅
  for i ← 1 to n
      minw_i ← ∞
      for j ← 1 to k
          w_ij ← Σ_{u∈S_j} w(uv_i)        (∗)
          if minw_i < w_ij
              minw_i ← w_ij
              minj ← j
      S_minj ← S_minj ∪ {v_i}
  return S_1, S_2, ..., S_k
```

This algorithm runs in $O(V+E) = O(n^2)$ time; each edge in G is considered exactly twice in the innermost loop $(*)$.

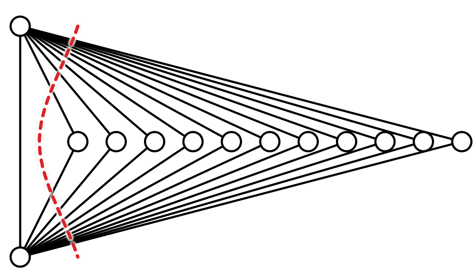
Let OPT denote the cost of the optimal partition, and let W denote the total weight of all edges in the graph. Clearly $OPT \le W$. ApproxMaxKCut computes a partition with cost $W - \sum_i minw_i$. For each i, we have $minw_i \ge \sum_j w_{ij}/k$, so

$$\sum_i minw_i \ge \sum_i \sum_j \frac{w_{ij}}{k} = \frac{W}{k}.$$

Thus, our algorithm computes a partition with cost $(1-1/k)W \ge (1-1/k)OPT$. ■

(b) Now suppose we wish to minimize the sum of the weights of edges that do *not* cross the partition. What approximation ratio does your algorithm from part!(a) achieve for this new problem? Justify your answer.

Solution: The approximation ratio can be arbitrarily bad, even when $k = 2$ and every edge has unit weight. Consider the graph obtained by adding a single edge to the complete bipartite graph $K_{2,n-2}$:



The optimal partition has cost 1—one subset contains the two leftmost vertices; the other subset contains everything else. If ApproxMaxKCut considers the two leftmost vertices first and second, it will assign them to different subsets. No matter how the other $n-2$ vertices are split up, the resulting partition has cost $n-2$. ■

Solution: The approximation ratio can be arbitrarily bad, even for graphs of constant size when $k = 2$. Consider a triangle with three vertices x, y, z, where $w(xy) = 1$ and $w(yz) = w(xz) = W$. The optimal partition is $\{x, y\}, \{z\}$, which has cost 1. If the greedy algorithm considers x and y before z, it will put x and y on different sides of the partition, which means the computed partition has cost W. ■

2. (a) Prove that NEXTFIT uses at most twice the optimal number of bins.

Solution: When the algorithm terminates, we have $Total[i-1] + Total[i] > 1$ for all $1 \le i \le bins$; otherwise, the algorithm would have put more items into bin $i-1$. Thus, $\sum_{i=1}^{bins} Total[i] \ge \lceil bins/2 \rceil$. On the other hand, we trivially observe that

$$OPT \ge \sum_{j=1}^{n} W[j] = \sum_{i=1}^{bins} Total[i].$$

It follows that $OPT \ge \lceil bins/2 \rceil$, or equivalently, $bins \le 2 \cdot OPT - 1$. ■

(b) Prove that FIRSTFIT uses at most twice the optimal number of bins.

Solution: As in part (a), we trivially observe that

$$OPT \ge \sum_{j=1}^{n} W[j] = \sum_{i=1}^{bins} Total[i]$$

when the algorithm terminates. At the end of the algorithm, there is at most one index i such that $Total[i] \le 1/2$. (If $Total[i] \le 1/2$ and $Total[j] \le 1/2$ for some $i < j$, then the first item placed into bin j could have been put into bin i instead.) Thus, $\sum_{i=1}^{bins} Total[i]$ is *strictly* larger than $(bins-1)/2$. Thus, $OPT > (bins-1)/2$; since OPT is an integer, it must be at least $bins/2$. ■

(c) ***[Extra Credit]*** Prove that if the weights are initially sorted in decreasing order, then FIRSTFIT uses at most $(4 \cdot OPT + 1)/3$ bins, where OPT is the optimal number of bins.

Solution: Recall that the items are sorted in decreasing order. We first prove the claims given in the exercise as follows:

- Suppose the kth item is the first to be placed in bin $OPT+1$. For purposes of proving a contradiction, suppose $W[k] > 1/3$. At this point of the algorithm, each of the first OPT bins contains at most two items. Moreover, for some j, the first j bins each contain exactly one item, which has size greater than $1/2$, and the rest each contain two items, each of size at least $1/3$. (Suppose bin x contains two items x_1 and x_2, and bin y contains only one item y_1, for some $x < y$. Then $W[x_1] \ge W[y_1]$ and $W[x_2] \ge W[k]$, so $W[y_1] + W[k] \le W[x_1] + W[x_2] \le 1$, which implies that item k could be placed into bin y instead of starting a new bin.)

  Now we claim there is no way to put the first k items into OPT bins. Let $A = \{1, 2, \ldots, j\}$ and $B = \{j+1, \ldots, k-1\}$. No bin can contain two items from A, since they each have weight greater than $1/2$. FIRSTFIT failed to place any item in B into one of the first j bins, so no bin can contain both an item from A and an item from B. Since each item in B has weight greater than $1/3$, no bin can contain more than two items in B. Finally, since FIRSTFIT placed exactly two items from B in each of the last $OPT - j$ bins, there must be exactly $2(OPT - j)$ items in B. Thus, *any* solution for $A \cup B$ uses at least OPT bins, with no room left over for the kth item. In other words, the optimal solution for the fist k items requires at least $OPT + 1$ bins. But this contradicts the *definition* of OPT!

  Thus, $W[k] \le 1/3$. We conclude that every item with weight greater than $1/3$ is placed in one of the first OPT bins.

- Call the items that FirstFit places outside the first *OPT* bins *extra* items. Let $x_1, x_2, \ldots$ be the extra items. For each i we must have $Total[i] + W[x_i] > 1$ for each i, since otherwise item x_i could have been placed in bin i. If there are *OPT* or more extra items, we can derive a contradiction as follows:

$$\begin{aligned} OPT &\geq \sum_{i=1}^{n} W[i] \\ &\geq \sum_{j=1}^{OPT} Total[j] + \sum_{i=1}^{OPT} W[x_i] \\ &= \sum_{i=1}^{OPT} (Total[i] + W[x_i]) \\ &> OPT \end{aligned}$$

  We conclude that FirstFit places at most $OPT - 1$ items outside the first *OPT* bins.

Finally, there are at most $OPT - 1$ extra items, each of size at most $1/3$, so there are at most $\lceil (OPT - 1)/3 \rceil \leq (OPT + 1)/3$ extra bins. Thus, the total number of bins is at most $(4 \cdot OPT + 1)/3$. ■

3. (a) Prove that the greedy algorithm for TSP has an approximation ratio of $O(\log n)$.

Solution: Let H be the Hamiltonian cycle computed by the greedy algorithm; for purposes of analysis, suppose H is directed in the order in which vertices were added to the greedy tour. Let $e_1, e_2, \ldots, e_n$ be the edges of H in order by *decreasing weight*, and for each index i, let v_i be the tail vertex of e_i; that is, edge e_i goes from the corresponding vertex v_i to some other vertex v_j. Finally let $w(e)$ denote the weight of any edge e.

Consider the vertices $v_1, v_2, \ldots, v_k$ with the k most expensive outgoing edges in T, and let OPT_k denote the cost of the optimal tour of just those k vertices. For all indices $i < j \le k$, we have $w(v_i v_j) \ge w(e_i) \ge w(e_k)$, since otherwise the greedy algorithm would have chosen $v_i v_j$ instead of e_i. It follows that $OPT_k \ge k \cdot w(e_k)$. On the other hand, we have $OPT_k \le OPT$, because the optimal tour of *all* vertices is also a tour of $v_1, v_2, \ldots, v_k$.

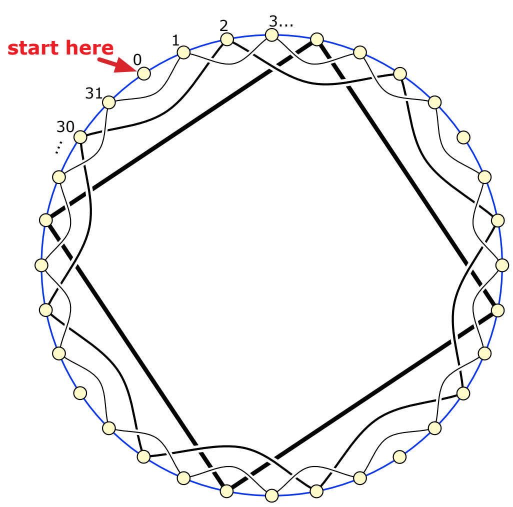
We conclude that the total weight of H is

$$\sum_{i=k}^{n} w(e_k) \;\le\; \sum_{i=k}^{n} \frac{OPT_k}{k} \;\le\; OPT \cdot \sum_{i=k}^{n} \frac{1}{k} \;=\; OPT \cdot H_n \;=\; OPT \cdot O(\log n),$$

where H_n is the nth harmonic number. ■

*(b) ***[Extra Credit]*** Prove that the approximation ratio for this algorithm is $\Omega(\log n)$.

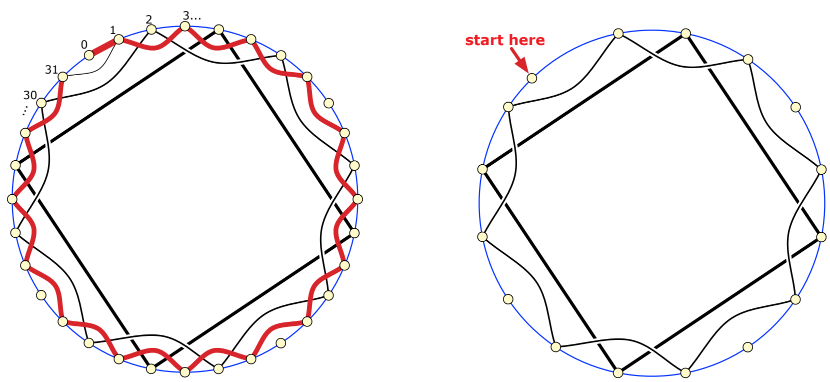
Solution: For any integer $k \ge 2$, we define a weighted graph G_k with $n = 2^k$ vertices as follows. We start with the *base cycle* C_n, giving each edge weight 2; we label the vertices $0, 1, 2, \ldots, n-1$ in order around the cycle. Then for each integer $0 \le i \le k-3$, we introduce a cycle of *shortcut* edges of weight 2^i through the odd multiples of 2^i, in order around the base cycle. Each shortcut cycle has *total* weight $n/2$.



The graph G_5. The outer circle is the base cycle C_{32}; the other edges are the shortcut cycles.

The ***metric completion*** G_k^* of G_k is defined as the weighted complete graph with the same vertices as G_k, where the weight of any edge uv is the shortest path distance from u to v in G_k. The metric completion of *any* weighted graph obeys the triangle inequality. Each shortcut edge uv in G_k has exactly 1/4 the length of the shortest path in C_n between u and v. Any path in G_k between adjacent vertices in C_n must traverse at least one edge in C_n. Thus, every edge in G_k is a shortest path between its endpoints; equivalently, every edge in G_k appears in the metric completion G_k^* with the same weight.

Any pair of vertices $2i$ and $2i+2$ has shortest path distance 2 in G_k. Thus, if we delete all the odd vertices from G_k, divide both the remaining vertex labels and remaining edge weights in half, and add a cycle of length $n/2$ connecting the even vertices in order, the resulting graph is precisely G_{k-1}. ***Thus, removing all the odd vertices from G_k^* and halving the remaining edge weights gives us G_{k-1}^*.*** This recursive structure is the key to the approximation analysis.



Middle: The first phase of the greedy TSP path in G_5.
Right: Removing the vertices of the first phase essentially leaves G_4.

Now suppose we run the nearest-neighbor algorithm on G_k^*, for some $k > 2$, starting at vertex 0. The nearest neighbors of vertex 0 are vertices 1 and $n-1$; assume without loss of generality that the algorithm chooses vertex 1. The nearest neighbors of any odd vertex are its adjacent odd vertices, so the algorithm visits all the odd vertices in (say) increasing order, traversing all but one shortcut edge of length 1 and ending with vertex $n-1$.

At this point, the state of the algorithm is almost exactly the same as if the original input were G_{k-1}^*; the only difference is that one of the edges leaving the start vertex is shorter than the other. (See the figure on the next page.) It follows by induction that the greedy algorithm traverses all but one edge in *every* shortcut cycle in G_k. (The base case $k=2$ is vacuous.) Thus, the total length of the resulting Hamiltonian cycle is at least

$$\sum_{i=0}^{k-3}(n/2-2^i) = (k-2)n/2-(2^{k-2}-1) = (n/2)\lg n - 5n/4 + 1 = \Omega(n\log n).$$

The graph G_k^* has a Hamiltonian cycle of length $2n$, namely the base cycle C_n. We conclude that the nearest-neighbor algorithm outputs an $\Omega(\log n)$-approximation. ■