Problem 1: MaxCut for trees:

HLD:

The key point is we need to find a partition of the nodes V into two sets of **equal** size, otherwise, we could put nodes in level one in set A and level two in set B and level three in set A and so on. In this problem, we need to associate nodes with numbers in order to obtain half of the number of nodes. We define $n = |V|$ and there is a weight w_{uv} for edge $(u, v) \epsilon E$.

We denote the root as r and two sets A and B. So the problem is that we need to find $n/2$ nodes from V such that $\sum_{u \epsilon A, v \epsilon B, (u,v) \epsilon E} w_{uv}$ is maximized.

Since we are given a binary tree $T = (V, E)$, each node in V has a subtree. We show that node u has a subtree T_u with n_u nodes. Here we need to use Dynamic Programming (Memorization) to record the total weight with respect to number of nodes.

We use $OPT(u, k)$ to denote the maximum cut of subtree T_u with a set size k, which is, choose $(k-1)$ nodes from subtree T_u together with node u to form a set while other nodes in subtree T_u form another set, and the cut between the two sets is the largest. We need to note that $k \leq n/2$ since the final set should have a size of $n/2$, so we do not need to consider set larger than $n/2$.

We do memorization from bottom to top, which is to compute the problem from the leaves. For a leaf u, it has no children, so $k = 1$(only include leaf u). Since there is no subtree (no edges), we have $OPT(u, 1) = 0$ for all leaves.

For node u with only one child v, we need to consider if node u and node v are in the same set. If they are in the same set, then $OPT(u, k) = OPT(v, k-1)$ since edge (u, v) is not a cut. If they are in different set, then $OPT(u, k) = w_{uv} + OPT(v, n_v - k + 1)$ since the set with node u inside will choose another $(k-1)$ nodes from subtree T_v, that is, set with node v will have $n_v - (k-1)$ nodes. We have:

$$OPT(u, k) = \max\left(OPT(v, k-1),\ w_{uv} + OPT(v, n_v - k + 1)\right)$$

For node u with two children: left child v and right child w, we have to choose m_1 nodes from the left subtree and m_2 nodes from the right subtree such that $m_1 + m_2 = k - 1 (0 \leq m_1 \leq k-1, 0 \leq m_2 \leq k-1)$. And for each child, we need to

decide whether it belongs to the same set as its parent u like described above. So the equation for this situation is:

$$OPT(u,k) = max_{0\le m_1\le k-1,\, m_2=k-1-m_1}(\max\big(OPT(v,m_1),\ w_{uv} + OPT(v,n_v-m_1)\big) + \max\big(OPT(w,m_2), w_{uw} + OPT(w,n_w-m_2)\big))$$

We compute the values from the bottom level to the root r. And return the answer $OPT(r,n/2)$.

Proof of Correctness:

We should note that $OPT(r,n/2)$ is the maximum weight of the cut between two sets of equal size. Since there are only two sides, root r must be in one of them, let's say, set A. Set A has another $(\frac{n}{2}-1)$ nodes selected from $r's$ left children and $r's$ right children. Since our computation ensures that for each step we obtain the optimal result for node u with respect to certain number, the result we returned will be the optimal one. From the last step, we choose from root $r's$ left subtree and right subtree, so we are basically selecting from all nodes to form two set with size $n/2$.

Other detailed explanation is in High-Level Description part.

Time Complexity:

There are n nodes and $k \le n/2$. Therefore we need to compute $O(n^2)$ OPTs. And for each OPT, we need to do at most $n/2$ comparison. So the total time complexity is $O(n^3)$.

Problem 2: Heaviest first

1.Recall that in a graph $G = (V,E)$, we say a set of nodes $S \subseteq V$ is independent if no two nodes in S are joined by an edge. In this question, for each node $v \in T$, either $v \in S$, or v must be deleted when one of its neighbor is selected by the greedy algorithm, otherwise, the while-loop won't terminate. If v is deleted because its neighbor v' is chosen, then $w(v) \le w(v')$. Since in one iteration, only the node of the maximum weight will be picked.

2. Define the optimal independent set with the maximum total weight is S_{opt}. We need to prove: $\frac{S}{S_{opt}} \ge \frac{1}{4}$.

First, we split these two sets into a few parts:

$$S_1 = \{v | v \in S, v \in S_{opt}\}$$
$$S_2 = \{v | v \in S_{opt}, v \notin S\}$$

$$S_3 = \{v | (u, v) \in E, u \in S_{opt}\}$$
$$S_4 = \{v | v \notin S_1, v \notin S_3, v \in S\}$$

We have $S_{opt} = S_1 + S_2$ which is easy to find and $S = S_1 + S_3 + S_4$ because $S_1 \cap S_3 = \emptyset$. We now prove it by contradiction. If there is a node v exists in both S_1 and S_3, then $v \in S_{opt}$ and at least one of its neighbor is in S_{opt} which contradicts that S_{opt} is an independent set. So we proved $S_1 \cap S_3 = \emptyset$.

Based on question 1, we have, for a node v in S_2, there must be a node $v' \in S$ so that $w(v) \leq w(v')$ and (v, v') is an edge of G. One node in S_3 can at most have 4 neighbors in S_2, so we have $4w(S_3) \geq w(S_2)$. Therefore, we have:

$$\begin{aligned} w(S) &= w(S_1) + w(S_3) + w(S_4) \\ &\geq w(S_1) + w(S_3) \\ &\geq w(S_1) + \frac{1}{4} w(S_2) \\ &\geq \tfrac{1}{4} w(S_1) + \tfrac{1}{4} w(S_2) \\ &= \frac{1}{4}\big(w(S_1) + w(S_2)\big) \\ &= \frac{1}{4} w(S_{opt}) \end{aligned}$$

From the process, we prove that the "heaviest-first" greedy algorithm returns an independent set of total weight at least ¼ times the maximum total weight of any independent set in the grid graph G.

Problem 3: Bin Packing

1.Since we assume that W is at least as large as each individual w_i, we have the input to this problem:

$$n + logW + \sum_{i=1}^{n} logw_i \leq n + logW + nlogW$$

So we only need to prove the merging heuristic terminates in time polynomial in $(n + logW)$. As we see from the merging algorithm, in each iteration, we will at most do $\binom{n}{2} = O(n^2)$ addition and then compare with bin size W. Since we should account for the time required to perform arithmetic operations, we cannot treat addition and comparison to be constant time. Since $w_i \leq W$ and we only merge bins with the union of their content no larger than W, we can limit each arithmetic operation within $O(logW)$ time.

There should be at most n iterations since in each loop we will drop one bin and the dropping operation may cause a displacement of all bins which takes $O(n)$ time. So the total time complexity is:

$$O((n^2 logW + n) \cdot n) = O((logW + n)^4)$$

Thus the merging heuristic terminates in time polynomial in the size of the input.

2. Assume we have 4 items with weights 1,2,5,5 and bins with size $W = 7$. According to the merging heuristic, the first two items may firstly be merged and then no two bins can be merged anymore. So we end up using 3 bins with weight (3,5,5). But the minimum number of bins should be 2 which contains (1+5, 2+5).

3.We define S to be the number of bins returned by the merging heuristic and S_{opt} be the minimum number of bins used to contains the items. We need to prove $\frac{S}{S_{opt}} \leq 2$.

We define total item weight $T = \sum_1^n w_i$. Since weight of items in each bins is no larger than W, we have $T \leq S \cdot W$ and $T \leq S_{opt} \cdot W$.

Now we need to find an upper bound for nominator S. We denote m as the minimum load of bins returned by the merging heuristic. If $m \leq W/2$, we have all the other $(S-1)$ bins whose load must be larger than $(W-m)$, otherwise, there still be two bins waiting to be merged. So we have:

$$\begin{aligned} T &\geq (W-m)(S-1) + m \\ &= WS - W - (S-2)m \end{aligned}$$

Here we only consider $S \geq 2$ since if $S = 1$, then minimum bin number is 1, the ratio becomes 1 which is larger than ½. Since $m \leq W/2$, we have:

$$\begin{aligned} T &\geq WS - W - (S-2)m \\ &\geq WS - W - (s-2)W/2 \\ &= WS/2 \end{aligned}$$

Since we have $T \leq S_{opt} \cdot W$, then $S_{opt} \cdot W \geq W \cdot S/2$. We obtain $\frac{S}{S_{opt}} \leq 2$.

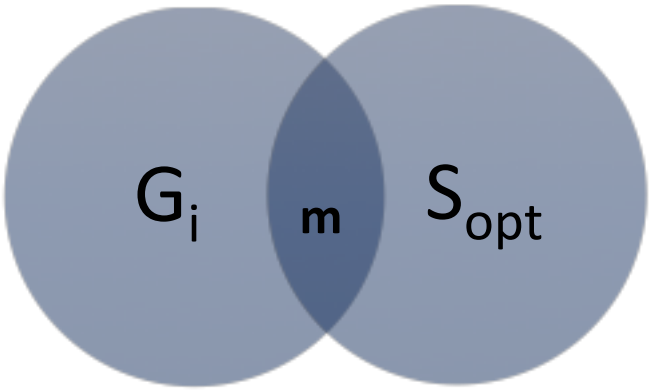
If $m \geq W/2$, then $T \geq S \cdot W/2$ which will lead to the same result as shown above. Above all, we show that the number of bins used in the packing returned by the heuristic is at most twice the minimum possible number of bins.

Problem 4: Maximum Coverage

Suppose the maximum collection of elements by picking k subsets is S_{opt} and the collection returned by the greedy algorithm is S_g. We define the set of covered elements from the greedy algorithm is G_i after iteration i and its corresponding weight is $w(G_i)$. We have $|G_0| = 0$ and $G_k = S_g$.

According to the greedy algorithm, we will choose the subset that could provide most additional weight. Let w_i be the additional weight gained during iteration i.

Recall that S_{opt} is comprised of k subsets. We denote set $X_i = S_{opt} - G_i$ as the set of elements that are not covered by the greedy algorithm. We should note that X_i could be covered by the k subsets that covered S_{opt}.



As shown in the figure above, G_i and S_{opt} share m same subsets $(0 \le m \le k)$. And the other $k - m$ subsets will cover more than or equals to X_i. So the weight of the $k - m$ subsets will be larger than $w(X_i)$. There must be a subset of these $k - m$ subsets has weight more than $\frac{w(X_i)}{k-m} \ge w(X_i)/k$. So we may choose a set with weight $w \ge w(X_i)/k$.

Therefore, we have:

$$\begin{aligned} w(G_i) &= w(G_{i-1}) + w \\ &\ge w(G_{i-1}) + w(X_i)/k \\ &\ge w(G_{i-1}) + (w(S_{opt}) - w(G_{i-1}))/k \\ &= \frac{w(S_{opt})}{k} + \left(1 - \frac{1}{k}\right) w(G_{i-1}) \end{aligned}$$

By iteration, we have:

$$w(G_k) \geq \frac{1}{k} * w(S_{opt})[1 + \left(1 - \frac{1}{k}\right) + \left(1 - \frac{1}{k}\right)^2 + \cdots + \left(1 - \frac{1}{k}\right)^{k-1}]$$
$$= w(S_{opt})\left(1 - \left(1 - \frac{1}{k}\right)^k\right)$$

So we get:

$$\frac{w(G_k)}{w(S_{opt})} \geq 1 - (1 - \frac{1}{k})^k$$

Now we deal with $1 - \left(1 - \frac{1}{k}\right)^k > 1 - \frac{1}{e}$, or equivalently, $kln\left(1 - \frac{1}{k}\right) < -1$ by taking logarithm. We rewrite is as $ln\left(1 - \frac{1}{k}\right) + \frac{1}{k} < 0$.

We define a function $f(x) = \ln(1 - x) + x$. We have $f(0) = 0$ and $f' = -\frac{1}{1-x} + 1$. Since $k \geq 1$, we have $0 < x \leq 1$. The max value of $f(x)$ is 0 when $0 < x \leq 1$. So we have $1 - \left(1 - \frac{1}{k}\right)^k > 1 - \frac{1}{e}$.