- **Algorithm**

  First, let $X_2 \subseteq X$ be the subset of elements of $X$ that are greater than $\frac{t}{k+1}$. Let $S$ be the set consisting of the subset that achieves the largest sum no greater than $t$ within $X_2$. Now sort the remaining elements of $X - X_2$ in nonincreasing order. For each element in this list, add it to $S$ if doing so would not cause the sum to exceed $t$.

- **Time Complexity**

  Every element in the subset $X_2$ is greater than $\frac{t}{k+1}$, so $S$ of the selected elements must contain less than $k$ elements. Therefore choosing the elements for $S$ by brute force search requires at most $O(\mid X \mid^k)$ time, which is polynomial to the input size $\mid X \mid$.

- **Proof**

  Suppose we are given a SUBSET-SUM instance $x$ of a set $X$ of integers and a desired sum $t$.

  Both the approximation solution and an exact solution will give the answer of $C(x) = c$ in the following cases:

  - If the sum $c$ of all integers in $X$ is smaller than $\frac{k}{k+1}t$;
  - If $X - X_2$, either empty or non-empty, fails to make the sum of $S$ to exceed $\frac{k}{k+1}t$.

  In these cases, the approximation gives an optimal solution. Otherwise, the approximation algorithm gives an answer $C(x)$ ranging anywhere from $L(x)$ up to $U(x)$.

  The upper bound $U(x)$ is $t$ in a best case when there are some numbers in $X$ that sum up to exactly $t$.

  The lower bound $L(x)$ is $\frac{k}{k+1}t$ because we are always able to find a subset with a sum between $\frac{k}{k+1}t$ and $t$ by keeping adding numbers until the sum is about to exceed $\frac{k}{k+1}t$. Suppose with one more number $x$ the sum will exceed $t$, then the sum before adding $x$ must be greater than $\frac{k}{k+1}t$ because $x$ is at most $\frac{t}{k+1}$.

- According to the above analysis,

  $$\frac{k}{k+1}t \leq C(x)$$

  and

  $$C^*(x) \leq t.$$

  Therefore,

  $$1 \leq \frac{C(x)}{C^*(x)} < \frac{t}{\frac{k}{k+1}t} = \frac{k+1}{k}.$$

  In other word, the algorithm has an approximation ratio of $\frac{k+1}{k}$.