Tarea 1

Saul Ivan Rivas Vega

Diseño y análisis de algoritmos

Equipo Completo:
Yadira Fleitas Toranzo
Diego de Jesús Isla Lopez
Saul Ivan Rivas Vega

28 de agosto de 2019

1. Ejercicio 1. Historia de Guerra : Psychic Modeling.

1.1. Describe el problema y la forma de resolverlo en tus propias palabras

1.1.1. Problema

El problema consiste en que en una lotería donde los tickets son de la forma:

$$T = \{t_1, t_2, t_3, ..., t_k\} \tag{1}$$

En el ejemplo del libro $k = 6$ y cada t_i es un número del conjunto:

$$A = \{1, 2, 3, ..., a\} \tag{2}$$

En el ejemplo $a = 44$, ademas ganar un premio se logra si del ticket ganador al menos l números coinciden en alguno de los tickets comprados, en el ejemplo $l = 3$, lo interesante es cuando se supone correcta la predicción de un psíquico que propone un subconjunto de A :

$$S = \{1, 2, 3, ..., n\} \tag{3}$$

En el ejemplo $n = 15$, pero ese conjunto no dice nada realmente, la verdadera aportación del psíquico es que de ese subconjunto S habrá a su vez otro subconjunto:

$$W = \{x_1, x_2, x_3, ..., x_j\} \tag{4}$$

Donde cada x es un elemento de S y para el ejemplo $j = 4$, esta asegurado, según el psíquico, que estos j números aparecerán en el tickets ganador.

Ahora si, el problema, dada estas definiciones ¿Cuál es la cantidad mínima (o al menos mas pequeña que la de el programa de la competencia) de tickets que se debe comprar para asegurar un premio?

1.1.2. Forma de resolverlo

Los pasos principales son generar los posibles subconjuntos de tamaño l del conjunto S, de ahí buscar un conjunto de tickets tal que no importa que subconjunto salga en el ticket ganador, nosotros ya tendremos al menos un

ticket que pueda ganar premio, es decir que nuestro conjunto de tickets cubre todos los posibles subconjuntos de tamaño l del conjunto S. Definamos pues el procedimiento.

- Generamos el conjunto L formado por todos los subconjuntos $\binom{n}{k}$ que son todos los tickets que podemos comprar dado el conjunto S.
- Generamos todos los subconjuntos $\binom{n}{l}$ que son los que debemos cubrir con nuestro subconjunto de tickets que compremos.
- Debemos contar con alguna función de ranking\unranking de tal forma que podamos etiquetar los subconjuntos $\binom{n}{k}$ y a los subconjuntos $\binom{n}{l}$.
- Debemos llevar registro de los subconjuntos $\binom{n}{l}$ que llevamos cubiertos para decidir si compramos el siguiente ticket que podamos comprar. Logrado a través de un bit-vector donde tenemos 0 si no esta cubierto o 1 en caso contrario.
- Para marcar de un ticket T los subconjuntos $\binom{n}{l}$ que cubre, se generan los subconjuntos $\binom{T}{l}$ y se marcan en el bit-vector.
- Para n lo suficientemente pequeño se puede buscar exhaustivamente en todos los posibles subconjuntos de tickets, es decir $\mathcal{P}(L)$ y para los demás se pueden usar técnicas aleatorias para la selección del subconjunto de tickets y quedarse con el mas pequeño.

Ahora bien, este fue el primer razonamiento al problema y según la historia el director de la lotería seguía obteniendo mejores resultados y revisando se dieron cuenta que en efecto este procedimiento no les daba subconjuntos lo suficientemente pequeños. La modificación para que si lo haga yace en el marcado ya que erróneamente pensaban que era necesario cubrir absolutamente todos los subconjuntos $\binom{n}{l}$ sin embargo no es necesario ya que si alguno no se cubre al completar el ticket para que tenga la longitud k existirá un subconjunto que sí cubramos.

- Al seleccionar un ticket T este cubre mas subconjuntos $\binom{n}{l}$ puesto que si se complementa cada uno de los subconjuntos $\binom{T}{l}$ para que tengan longitud k habrá otros subconjuntos $\binom{n}{l}$ que adicionalmente cubrirá.
- Un punto adicional a considerar puede ser evaluar que tan valiosa es la información del psíquico con respecto a j pues si es igual o muy cercano

a l realmente no hay mucho en donde considerar a j, sin embargo si es mayor podemos realizar una búsqueda en $\binom{n}{j}$ la cual puede ser menor a $\binom{n}{l}$ teniendo un menor conjunto de elementos a cubrir, y cada ticket cubriría un mayor numero de subconjutos.

1.2. Explica el método LottoTicketSet

El método LottoTicketSet recibe como argumentos n, k y l, y sigue los siguientes pasos:

1. Se inicializa el bit-vector de tamaño $\binom{n}{l}$ al principio todo en 0 o false pues no hemos marcado ningún subconjunto todavía.

2. Mientras exista un elemento sin marcar en V, este proceso termina cuando hemos cubierto los subconjuntos para ganar al menos un premio.

   - Se selecciona un subconjunto T de S como el siguiente ticket a comprar.
   - Se generan todos los subconjuntos T_i de $\binom{T}{l}$ y para cada uno con la función $rank()$ obtenemos su etiqueta y la marcamos en V.

3. Reportamos el conjunto de tickets comprado.

El método ignora como es que se busca exhaustivamente la respuesta, es más bien la definición de como llevar el registro de los subconjuntos $\binom{n}{l}$ que el actual conjunto de tickets cubre, el llamado *book-keeping*.

1.3. Explica a qué se refiere con implementar un algoritmo de fuerza bruta basada en este método; describe su complejidad explicando si es polinomial o no.

Para el algoritmo de fuerza bruta para generar los posibles subconjuntos de tickets consideremos los siguientes datos:

- Un conjunto L conformado por los posibles subconjuntos de tamaño k de S, es decir L es el conjunto de todos los posibles tickets a comprar .

- El conjunto U que es el conjunto potencia de L, es decir $\mathcal{P}(L)$.

Si vamos a buscar exhaustivamente por todos los subconjuntos de tickets nos estamos refiriendo a que debemos buscar en todo U. Ahora si nos tomara al menos $|U|$ y como $U = \mathcal{P}(L)$ entonces $|U| = 2^{|L|}$ y a su vez $|L| = \binom{n}{k}$. Terminamos entonces que tenemos al menos que hacer $2^{\binom{n}{k}}$ operaciones dados un conjunto de n números para seleccionar tickets de longitud k. Al ser exponencial el crecimiento de esta búsqueda con respecto al tamaño de la entrada se determina que no es polinomial pues no hay polinomio que crezca al menos tan rápido como $2^{\binom{n}{k}}$.

1.4. Explica a qué se refiere con que la historia tiene un final feliz.

La historia tiene un final feliz por que su procedimiento de búsqueda y casi toda su solución no fue en vano, la historia tendría un final triste si no resolvieran al final el problema. Solucionaron el problema al final modificando un poco su solución inicial pues como hemos revisado el problema era que no era necesario cubrir explícitamente todos los subconjuntos $\binom{n}{l}$, realizando las modificaciones como dicen, para registrar los subconjuntos $\binom{n}{l}$ que tienen el crédito de cubrir con el conjunto de tickets.

2. Ejercicio 2. Infinitos.

2.1. Considera el conjunto de todas las funciones de $\mathbb{N}$ a $\mathbb{N}$ (los naturales), demuestra que no es numerable.

Sea F el conjunto de todas las funciones de $\mathbb{N}$ a $\mathbb{N}$, ahora para demostrar que no es numerable hagamoslo por contradicción.

Primero supongamos que F es numerable, por lo tanto lo podemos ver de la forma:

$$F = \{f_1, f_2, f_3, ...\} \tag{5}$$

Donde un elemento f_i de F es una función de $\mathbb{N}$ a $\mathbb{N}$ e i es elemento de $\mathbb{N}$.

Ahora intentemos utilizando el argumento de la diagonalización de Cantor para definir una función que contradiga la suposición inicial. La técnica de la diagonalización consiste en formar un elemento externo que sea distinto a cada elemento del conjunto para demostrar por contradicción que el conjunto no lo contiene cuando se supuso en un principio que si lo contenía.

El elemento que formaremos es una función, una función que va de $\mathbb{N}$ a $\mathbb{N}$, y que sea distinta a cada f_i siendo evaluada en un i donde i es elemento de $\mathbb{N}$.

$$g(i) = f_i(i) + 1 \tag{6}$$

Ahora como g es una función que va de $\mathbb{N}$ a $\mathbb{N}$ y F contiene a todas las funciones que van de $\mathbb{N}$ a $\mathbb{N}$ y es numerable, según nuestra suposición inicial, significa que hay una función en F que sea la misma que g, pero la evaluación de $g(i)$ es distinta a f_i para cualquier $i \in \mathbb{N}$, esto implica una contradicción pues g debería ser parte y a la vez no de F, por lo tanto no es posible crear una lista numerable de todas las funciones de $\mathbb{N}$ a $\mathbb{N}$, así F es no numerable.

2.2. Por otro lado, un programa de computadora se puede ver como una cadena binaria finita; demuestra que el conjunto de todos los programas es numerable.

Si un programa se puede ver como una cadena binaria finita, probemos pues que el conjunto de cadenas binarias finitas es numerable. Para demostrarlo definamos una función inyectiva que pueda dar un valor único a cada cadena binaria finita por que de existir esta función podremos enumerar el conjunto de cadenas binarias finitas, demostrando que el conjunto de todas la cadena binarias finitas es numerable y de igual forma el conjunto de todos los programas.

Primero consideremos la siguiente función que devuelve la cantidad de distintas cadenas binarias de longitud i para cada $i \in \mathbb{N}$.

$$f(i) = 2^i \tag{7}$$

Ahora si modifiquemos esta función para que al evaluarla en i nos de la suma de las distintas cadenas binarias de cada longitud x donde $0 \leq x \leq i$.

$$f(i) = \begin{cases} 0, & \text{i} = 0 \\ 2^i + f(i-1), & \text{i} > 0 \end{cases} \tag{8}$$

Consideremos p una cadena binaria finita, p_i el valor en la posición i en p y $|p|$ su longitud. Definamos ahora una función que reciba como argumento una cadena binaria finita y al evaluarla devuelva el valor decimal de la cadena, que es a su vez un elemento en $\mathbb{N}$.

$$base10(p) = \sum_{n=0}^{|p|-1} 2^{|p|-n-1} \times p_n \tag{9}$$

Así evaluandola con 00101 obtendremos 5.

$$\begin{aligned} base10(00101) &= (2^{5-0-1} \times 0) + (2^{5-1-1} \times 0) + (2^{5-2-1} \times 1) \\ &\quad + (2^{5-3-1} \times 0) + (2^{5-4-1} \times 1) \\ &= (2^4 \times 0) + (2^3 \times 0) + (2^2 \times 1) \\ &\quad + (2^1 \times 0) + (2^0 \times 1) \\ &= 0 + 0 + 4 + 0 + 1 \\ &= 5 \end{aligned} \tag{10}$$

Tomando las equaciones 8 y 9 definamos una función que reciba como argumento una cadena binaria finita y devuelva un valor que sera único para cada cadena.

$$g(p) = base10(p) + f(|p| - 1) \tag{11}$$

La función g es inyectiva pues a cada cadena binaria finita le da un valor en $\mathbb{N}$ y ese valor no se repite para otra cadena binaria finita.

La existencia de la función g nos puede enumerar entonces un conjunto de cadenas binarias finitas y como el conjunto de todos los programas es un conjunto de cadenas binarias finitas se llega a que el conjunto de todos los programas es numerable.

2.3. Explica intuitivamente por qué el conjunto de todas las máquinas de Turing es numerable.

La definición formal de una máquina de Turing esta compuesta por objetos discretos, esto es que son conjuntos numerables como lo son su alfabeto, sus estados y transiciones, los cuales a su vez pueden ser codificados. Ahora bien si una maquina de Turing puede ser codificada en una cadena de un alfabeto numerable finito, posteriormente podemos representar dicha cadena en una cadena binaria finita. Entonces el conjunto de todas las maquinas de Turing pasa a ser un conjunto de todas las maquinas de turing codificadas y al final tenemos un conjunto de todas las maquinas de turing representadas como cadenas binarias finitas y como en el ejercicio 2.2 esta demostrado que el conjunto de todas las cadenas binarias finitas es numerable entonces el conjunto de todas las maquinas de turing numerable.

3. Ejercicio 3. Apareamientos estables

3.1. Considera el problema de roommates, para $2n$ personas.

3.1.1. Escribe un algoritmo similar al de Gale-Shapley, pero para este problema.

Data: Matriz de preferencias de las personas M de $n \times n$
Result: Un apareamiento E entre las personas
Inicializar una lista V con todos las personas, pues originalmente todos están sin pareja;
Inicializar como una lista vacia E pues todavía no hay parejas;
while *Haya algún elemento V_i en V* **do**
    Selecciona una persona p_j a la que mas prefiera V_i y con la que no haya intentado emparejarse antes;
    if *p_j esta en una pareja en E* **then**
        if *p_j tiene una mayor preferencia por V_i que su actual pareja* **then**
            Insertar a la pareja actual de p_j en V;
            Sacar de E la pareja en la que estaba p_j;
            Insertar la pareja de p_j con V_i en E;
            Sacar a V_i de V;
        else
            continuar con la siguiente iteración;
        end
    else
        Insertar la pareja de p_j con V_i en E;
        Sacar a V_i de V;
        Sacar a p_j de V;
    end
end
Reportar la lista de parejas E;

Algorithm 1: Algoritmo para el problema de los $2n$ roommates

3.1.2. Demuestra que el algoritmo del inciso anterior siempre termina y calcula su complejidad.

Una persona p_i permanecerá en la lista V si es que no tiene pareja, en cada iteración seleccionara a otra persona p_j con la que no haya intentado emparejarse antes y tenga la mayor preferencia. Esto significa p_i puede buscar emparejarse a lo más $n-1$ veces pues sería el caso donde le pregunto a todas las demás $n-1$ personas. El algoritmo termina pues después de que las n personas hayan intentado emparejarse $n-1$ veces no quedará ninguna otra pareja que intentar, ahora si todas las personas intentaron emparejarse con todas las demás significa que estan emparejadas con la última mejor opción que intentaron así el algoritmo siempre termina, no siempre el emparejamiento final es estable.

La complejidad del algoritmo seguiría de ese peor caso donde las n personas hayan intentado emparejarse $n-1$ veces tendríamos a lo mas $n(n-1)$ operaciones. En notación O tendríamos que el algoritmo es de orden $O(n(n-1))$ que lleva a $O(n^2-n)$ y finalmente $O(n^2)$.

3.1.3. Al terminar, ¿el algoritmo encuentra un apareamiento completo?

Si, puesto que como se menciona en el ejercicio anterior, al terminar el algoritmo en el peor caso las n personas han intentado emparejarse $n-1$ veces, lo que es intentar con todas las demás personas, si una persona se quedo sin pareja significa que hay otra sin pareja con la que no intento emparejarse, sin embargo si intento emparejarse con todas las demás, por lo tanto cada persona terminará con una pareja.

3.1.4. Explica qué sucede cuando se corre sobre una lista de preferencias para 4 personas que no tiene un apareamiento estable (con un ejemplo).

	1st	**2nd**	**3rd**
A	B	C	D
B	C	A	D
C	A	B	D
D	A	B	C

Cuadro 1: Matriz de preferencias de las personas.

1. Insertamos A, B, C y D a la lista V.
2. Inicializamos E como una lista vacía.
3. Tomamos a A de V
4. Intentamos emparejar A con su mejor opción con la que no haya intentado emparejarse, en este caso B.
5. B no tiene pareja por lo tanto sacamos a A y a B de V e insertamos la pareja $A - B$ en E.
6. Tomamos a C de V
7. Intentamos emparejar C con su mejor opción con la que no haya intentado emparejarse, en este caso A.
8. continuamos con la siguiente iteración por que A tiene pareja y no prefiere a C
9. Tomamos a C de V
10. Intentamos emparejar C con su mejor opción con la que no haya intentado emparejarse, en este caso B.
11. B tiene pareja y como prefiere a C entonces insertamos A en V, sacamos la pareja $A - B$ de E, insertamos la pareja $C - B$ en E y sacamos a C de V

12. Tomamos a D de V

13. Intentamos emparejar D con su mejor opción con la que no haya intentado emparejarse, en este caso A.

14. A no tiene pareja por lo tanto sacamos a D y a A de V e insertamos la pareja $D - A$ en E.

15. No hay mas elementos en V por lo tanto reportamos a E

$$E = \{C - B, D - A\} \tag{12}$$

El algoritmo termino sin embargo el apareamiento que dio aunque completo, es inestable pues existe la pareja $A - C$ donde A prefiere estar con C y a su vez C prefiere estar con A.

3.2. Escribe un algoritmo parecido al Gale-Shapely, pero basado en esta idea,

https://www.ece.ucsb.edu/ jrmarden/files/ece194v/lecture05.pdf En la que se repiten etapas como la que sigue: En cada etapa, todos los hombres libres mandan una propuesta a la primera mujer que prefieren y a la cual no han enviado propuesta. Cada mujer selecciona al preferido de las propuestas que ha recibido hasta el momento y lo pone en una lista de espera; a todos los demás les avisa que jamás se casará con ellos. Demuestra que tu algoritmo es correcto (encuentra un apareamiento estable completo). Demuestra que termina, explicando que:

1. Cada hombre es rechazado a lo mas n-1 veces

2. En cada etapa no-final, al menos uno es rechazado

3. Cada mujer recibe al menos una propuesta

4. El número máximo de etapas es 1+n(n-1)+1

Inicializar una lista R que representa a los hombres rechazados;
for *cada hombre* h_i **do**
  h_i le hará propuesta a la primera mujer en su lista de preferencias
end
for *cada mujer* m_i **do**
  if m_i *tiene mas de una propuesta* **then**
    Seleccionar a p_i quien es el favorito de m_i de entre todas sus propuestas y lo inserta en su lista de espera;
    m_i rechaza las demas propuestas informandoles que nunca se casará con ellos;
    se insertan a los hombres rechasados a la lista R;
  else
    m_i inserta al hombre de su única propuesta en su lista de espera;
  end
end
while *Haya algún hombre rechazado en* R **do**
  for *cada hombre rechazado* R_i *en* R **do**
    R_i se le propone a la siguiente mujer en su lista de preferencia que no lo haya rechazado;
  end
  for *cada mujer* m_i **do**
    if m_i *tiene mas de una propuesta* **then**
      Seleccionar a p_i quien es el favorito de m_i de entre todas sus propuestas y su lista de espera;
      m_i rechaza las demas propuestas informándoles que nunca se casará con ellos.;
      se insertan a los hombres rechasados a la lista R;
      if p_i *es distinto a quien estaba en su lista de espera* **then**
        m_i reemplaza al que esta en su lista de espera con p_i informandole a quien estaba que nunca se casara con él y se inserta a la lista R;
      else
      end
      if m_i *no tiene a nadie en su lista de espera* **then**
        m_i inserta a p_i en su lista de espera;
      else
      end
    else
      m_i inserta al hombre de su única propuesta en su lista de espera;
    end
  end
end
Reportar la lista de parejas dada por las listas de espera de cada mujer m_i;

Algorithm 2: Algoritmo por etapas.

3.2.1. Cada hombre es rechazado a lo mas n-1 veces

Demostración. por contradicción, supongamos que rechazan n veces a un hombre h_i eso significa que se le propuso a todas las n mujeres por que no se le propone a una mujer si esta ya lo rechazo, si fue rechazado por todas las n mujeres no tiene pareja, si un hombre no tiene pareja entonces una mujer no tiene pareja, si una mujer no tiene pareja significa que no recibió propuesta alguna puesto que una mujer se empareja si recibió al menos una propuesta, sin embargo esto contradice nuestra suposición inicial que es que h_i si le propuso a todas las mujeres, por lo tanto h_i no puede ser rechazado n veces o mas, se demuestra que a lo mas cada hombre es rechazado $n-1$ veces.

3.2.2. En cada etapa no-final, al menos uno es rechazado

Demostración. La etapa final consiste en una etapa en la que todos los hombres son aceptados por la mujer a la que se le propusieron, la única condición que se requiere para iniciar una nueva etapa es que haya al menos un rechazado por que son ellos los que harán nuevas propuestas, si ninguno es rechazado no habrá quien haga nuevas propuestas si no hay nuevas propuestas es un etapa final, entonces que no haya rechazados es suficiente para decir que es la etapa final y por lo tanto si hay al menos un rechazado quiere decir que es una etapa no-final.

3.2.3. Cada mujer recibe al menos una propuesta

Demostración. por contradicción, supongamos que una mujer no recibió ni una propuesta, eso significa que quedó sin pareja, eso significa que hay un hombre sin pareja, ahora un hombre sin pareja significa que fue rechazado por todas las n mujeres pero como se demostró anteriormente un hombre no puede ser rechazado mas de $n-1$ veces pues se le propuso a todas las n mujeres y al menos una lo acepto, sin embargo esto contradice nuestra suposición inicial de que una mujer no recibió ni una propuesta, por lo tanto no hay mujer que se quede sin propuesta alguna, finalmente esto significa que cada mujer recibe al menos una propuesta.

3.2.4. El número máximo de etapas es 1+n(n-1)+1

El número de etapas son:
Fase inicial: 1
Como demostramos que cada hombre es rechazado $n-1$ veces y e cada etapa al menos uno es rechazado podemos suponer que cada hombre de los n es rechazado $n-1$ veces: $n(n-1)$
Fase final: 1
Finalmente tenemos que hay: $1+n(n-1)+1$ etapas.

3.2.5. Demostrar que el algoritmo termina

Teniendo que cada hombre es a lo mas rechazado $n-1$ veces, en cada etapa no-final al menos uno es rechazado, cada mujer recibe al menos una propuesta y el número máximo de etapas es $1+n(n-1)+1$ podemos decir que el algoritmo termina pues tenemos demostrado que hay un numero finito máximo de etapas.

3.3. Describir el vídeo del premio nobel de economía 2012

El vídeo consiste en la ceremonia en la que otorgan el premio nobel de economía en el año 2012, los que recibieron el premio fueron Alvin E. Roth y Lloyd S. Shapley. En el año de 1962 David Gale y Shapley propusieron un algoritmo de aceptación diferida con el que probaron emparejamientos, sin embargo Roth realizó un trabajo de extensión al analizar las aplicaciones practicas en emparejamientos de mercados económicos, casos de elección de hospitales y doctores, o de escuelas y estudiantes. Mencionan también las distintas implicaciones de modificar las verdaderas preferencias de los participantes en un emparejamiento dado, sin embargo aunque mínimas estas consecuencias llevaron a que se realicen reportes de evaluaciones en mercados de emparejamiento con los distintos diseños económicos, como el realizado por Elliott Peranson y Alvin E. Roth.
En el vídeo también se realiza una demostración del algoritmo en el emparejamiento con el ejemplo de los hospitales y doctores. Finalmente los casos en donde se busca un emparejamiento estable y las consecuencias de parejas inestables son de particular sensibilidad como lo son hospitales y doctores o donadores de órganos y pacientes.

4. Ejercicio 4.

Demuestra que un algoritmo es de tiempo polinomial si tiene la siguiente propiedad de escalabilidad: Cuando el tamaño de la entrada se duplica, el tiempo de ejecución del algoritmo es, a lo más, incrementado por un factor de C, para alguna constante C.

4.1. Demostración

El enunciado describe que si el algoritmo tiene dicha propiedad de escalabilidad hay que demostrar que el algoritmo es polinomial.

Podríamos intentar demostrarlo por contradicción, es decir demostrar que si tiene la propiedad de escalabilidad es no-polinomial. Sin embargo entra la pregunta que cosa es "no-polinomial", lo cual presenta mas complicaciones al definir dicha separación entre polinomial y no-polinomial con solo la propiedad de escalabilidad.

Intentemos ver pues en que consiste que un algoritmo sea polinomial y que consiste tener la propiedad de escalabilidad descrita en el enunciado.

Algoritmo polinomial Un algoritmo es considerado de tiempo polinomial si es que existen una constante $c > 0$ y una constante $d > 0$ tal que para cualquier entrada de tamaño N su tiempo de ejecución este delimitado por cN^d operaciones.

Propiedad de escalabilidad Un algoritmo tiene la propiedad de escalabilidad descrita en el enunciado si se cumple que al duplicar el tamaño de la entrada el tiempo de ejecución sea incrementado a lo más por un factor C, para alguna constante C. Definamos como $f(n)$ al tiempo de ejecución del algoritmo. Entonces la propiedad de escalabilidad se vería como: $f(2n) = Cf(n)$

Demostración de que esta propiedad de escalabilidad esta presente en un algoritmo de tiempo polinomial

Definamos como $f(n)$ la función que describe el tiempo de ejecución del algoritmo.

$$
\begin{aligned}
f(n) &= cn^d \\
&\text{Donde } c \text{ y } d \text{ son constantes mayores a 0 .} \\
f(2n) &= c(2n)^d \\
&= c(2)^d(n)^d \\
&\text{La constante } C \text{ con la que creció al duplicar la entrada es } (2^d) \\
f(2n) &= (2^d)f(n) \\
&\text{Cumpliendo que:} \\
f(2n) &= Cf(n)
\end{aligned}
\tag{13}
$$

Ejemplo de que esta propiedad de escalabilidad no esta presente en un algoritmo de tiempo exponencial

Definamos como $f(n)$ la función que describe el tiempo de ejecución del algoritmo.

$$
\begin{aligned}
f(n) &= 2^n \\
f(2n) &= (2)^{2n} \\
&= (2^n)^2 \\
&= (2^n) \times (2^n) \\
&\text{Aquí el tiempo de ejecución no se incremento por una constante sino por } (2^n) \\
&\text{Lo cual es:} \\
f(2n) &= f(n) \times f(n) \\
&\text{Por lo tanto NO cumple que:} \\
f(2n) &= Cf(n)
\end{aligned}
\tag{14}
$$

Tomando en cuenta 14 podemos decir que si el algoritmo cumple la propiedad de escalabilidad no es exponencial, el tiempo de ejecución del algoritmo no incrementa tan rápido como una función exponencial, por lo tanto es polinomial.
En una segunda revisión la duda yace en que si el tiempo de ejecución de un algoritmo no crece tan rápido como una función exponencial y que todo algoritmo de tiempo polinomial tiene la propiedad de escalabilidad es suficiente para decir que si un algoritmo tiene la propiedad de escalabilidad es un algoritmo de tiempo polinomial.
Si consideramos que cualquier función que no crezca tan rápido como una exponencial es siempre polinomial entonces esa sería la demostración sin embargo dicha afirmación requeriría una mas definida linea divisora entre la clasificación de algoritmos basada en tiempos de ejecución.